UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Escola d'Enginyeria de Telecomunicació
i Aeroespacial de Castelldefels

# TREBALL DE FI DE GRAU

**TÍTOL DEL TFG: Sistemes de posicionament local indoor per a aplicacions dron**

**TITULACIÓ: Grau en Enginyeria d'Aeronavegació**

**AUTORS: Miquel Roger Fairman Faiman**
**Nil Seguer Hereter**

**DIRECTOR: Òscar Casas Piedrafita**

**DATA: 6 de febrer de 2017**

**Títol:** Sistemes de posicionament local indoor per a aplicacions dron

**Autors:** Miquel Roger Fairman Faiman
Nil Seguer Hereter

**Director:** Òscar Casas Piedrafita

**Data:** 6 de febrer de 2017

**Resum**

Aquest treball consta de l'estudi i implementació del control i el posicionament local *indoor* de drons.

Està dividit en tres parts. La primera tracta de l'estudi teòric dels sistemes actuals de posicionament, ja siguin globals com locals. S'expliquen els tipus de sistemes de posicionament locals i es fa una comparativa entre les diferents tecnologies disponibles, entre les que destaquen la radiofreqüència, el làser, el magnetisme i la visió per computador.

A continuació s'estudia i s'indaga sobre les plataformes en què es basa el treball, els drons Crazyflie de *software* obert per a la integració d'un sistema de posicionament local remot passiu i el Parrot AR.Drone pels sistemes actius. Es crea un sistema de control primari pel Crazyflie que inclou un sistema de seguretat enfront a caigudes i es descarta el Parrot per la impossibilitat de rebre les imatges capturades.

Per últim, s'estudia diferents sistemes de posicionament local, les seves característiques i la seva possible implementació i integració en el nostre estudi. S'escull com a sistema de posicionament local la visió per computador pel preu reduït i la facilitat d'implementació.

Mitjançant el llenguatge de programació Python, la llibreria lliure de visió per computador OpenCV i aprofitant el codi obert del *software* dels dos drons amb els que es treballa, s'ha dut a terme el control i posicionament.

De forma experimental s'ha intentat posicionar un dron Crazyflie dins d'un àrea d'interès mitjançant la visió per computador. No s'ha aconseguit però estabilitzar el dron dins de l'àrea més de 2 segons.

**Title :** Local positioning systems for dron applications

**Authors:** Miquel Roger Fairman Faiman
Nil Seguer Hereter

**Advisor:** Òscar Casas Piedrafita

**Date:** February 6, 2017

**Overview**

This document consists of the study and implementation of the control and the indoor local positioning of drones.

It is divided into three different parts. The first one is about the theoretical study of the global and local current positioning systems. The types of local positioning systems are described and they will be compared based on the different technologies they use such as radiofrequency, laser, magnetic and computer vision.

Then, it is studied and investigated the basis of the project, the open source drone Crazyflie that will be used for the remote passive location systems and Parrot AR.Drone for the remote active location system. A primary control system will be developed in order to fly the Crazyflie and this system will include a fall safe to protect the drone in case it falls.

Finally, some different local positioning systems have been studied as well as their characteristics and the possibility to implement and integrate them into our project. Computer vision has been chosen to deploy the system for being economically cheap and easy to implement.

Using the Python programming language, the open library OpenCV for computer vision and taking advantage of open source software of the two drones that we work with, the control and positioning have been carried out.

Experimentally, it has been tried to position the drones into a target area using computer vision. Unfortunately we have not been able to stabilize the Crazyflie for more than 2 seconds inside the desired area.

Agrair al director del projecte, l'Òscar Casas, per la seva predisposició a ajudar-nos a desenvolupar un bon projecte i per una actitud immillorable en tot moment.

D'altre banda agrair als nostres familiars i amics l'ajut mostrat durant aquests quatre/cinc anys que culminen amb aquest treball

# ÍNDEX

# ÍNDEX DE FIGURES

# ÍNDEX DE TAULES

# INTRODUCCIÓ

Durant els últims 70-80 anys un dels sectors tecnològics i d'oci que ha experimentat un creixement més pronunciat ha estat el del món aeronàutic. Aquest *boom* causat principalment per la creació i evolució d'aquest sector, ha permès el desenvolupament de moltes àrees directament lligades a l'aviació. Una d'aquestes, i la que més ens interessa a nosaltres com a sector d'estudi, és la dels drons.

Si bé és cert que en el món militar les tecnologies de control remot i de drons es van implementant des de fa unes quatre o cinc dècades, en el camp civil des del començament del segle vint-i-u el creixement de les tecnologies dron ha evolucionat notablement. El camp civil rep una atenció significativa per part de la comunitat d'investigació en robòtica. El motiu principal d'aquest creixement ha estat la creació d'instituts de recerca i les possibles millores multidisciplinars que aquestes tecnologies poden aportar en l'àmbit civil i també de l'oci. Entre aquestes aplicacions es poden destacar l'àmbit d'emergències, *monitorització*, comunicacions, serveis metereològics, l'agricultura, el sector industrial... De fet, no només poden aportar aplicacions com a actors, sinó que la seva pròpia tecnologia ja ha estat provada en diferents àmbits com pot ser el de l'automoció.

Els denominats UAVs (de l'anglès *Unmaned Aerial Vehicle*) també han evolucionat en aquest últims quinze anys, intentant adaptar-se a la legislació vigent per tal de garantir la integritat i seguretat de la població a les noves necessitats del mercat, així com als usuaris. Avui en dia aquestes tecnologies són a l'abast de tothom. És molt senzill aconseguir un dron i en molt poc temps dominar les tècniques requerides per executar un vol. Encara però, hi ha molts problemes a solucionar en aquest àmbit. L'objectiu principal d'aquest estudi és desenvolupar i buscar solucions a una problemàtica de les tecnologies dron: el posicionament local *indoor*.

En espais oberts, la navegació i control del dron es fa mitjançant sistemes de posicionament globals com podrien ser el GPS (EEUU), GLONASS (Rússia), GALILEO (Europa)... Però també es poden donar situacions en què la cobertura satèl·lit no és la idònia, o en espais tancats on el posicionament dels drons és més complicat i és necessari l'ús de diferents sistemes alternatius per tal de poder suplir el sistema de posicionament global.

Per tant, **poder detectar i posicionar un dron** on les condicions no són les més favorables per a sistemes de posicionament globals serà l'objectiu d'aquest estudi.

Arran d'aquesta missió principal sorgiran nous reptes com, per exemple: el control bàsic dels drons, la integració de sistemes de control i posicionament, les diferents alternatives de posicionament local, etc. Aquests s'hauran d'estudiar per tal de poder complementar el nostre treball. Un segon objectiu doncs, és **estudiar l'estabilitat i el control** dels drons escollits per a realitzar la part pràctica.

D'aquesta manera el nostre estudi estarà dividit en diferents parts ben diferenciades. Per una banda l'estudi previ que es tracta d'un treball pràcticament tot teòric sobre la informació necessària per conèixer quines bases hi ha fetes en el camp d'interès i per poder entendre i usar a posteriori els drons. A continuació, s'introdueixen diferents possibles mètodes com són la detecció de figures, colors o fins i tot moviment que podrien ser usats pel posicionament local *indoor*. Per acabar, es presenta la part pràctica del nostre estudi. En aquesta s'explica tant la preparació prèvia com l'execució de les proves executades amb els nostres drons. Per últim, s'analitzaran les dades i s'extrauran les conclusions

pertinents segons els resultats obtinguts en la part pràctica.

# CAPÍTOL 1. SISTEMES DE POSICIONAMENT

## 1.1. Sistemes de posicionament Globals

En l'actualitat el sistema de posicionament *de facto* és el GPS (*Global Positioning System*). El GPS funciona mesurant i triangulant les distàncies a un mínim de 4 satèl·lits per tal d'eliminar les possibles ambigüitats i determinar així, la posició geogràfica de qualsevol punt a tota la terra com s'aprecia a la figura 1.1

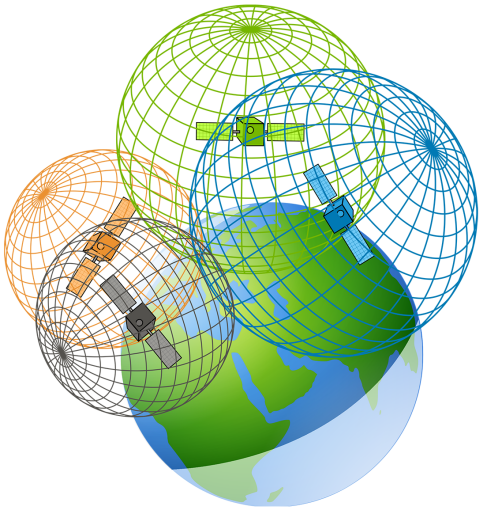

Figura 1.1: Esquemàtic del funcionament del GPS

L'origen del GPS al 1978 va ser militar [3] i més endavant, vora l'any 2000, es va obrir al sector civil amb certes limitacions. És a dir, el senyal GPS que es rep ha estat intencionalment degradat i per tant, no és tant precís com podria ser-ho. Malgrat aquest inconvenient, és suficientment precís per a la indústria dels drons en les zones obertes. A més a més, aquest posicionament no només es basa en les dades del GPS sinó que normalment també combina informació dels sensors inercials i dels acceleròmetres incorporats als receptors per determinar l'actitud i la possible posició en cas de pèrdua de la connexió GPS. L'exemple més senzill d'avui en dia podria ser quan es té el GPS del mòbil activat i s'entra en un túnel. A la pantalla del mòbil apareixerà com que no tenim cobertura GPS però es continuarà tenint la posició estimada gràcies al giroscopis i acceleròmetres que té el telefon. S'usarà doncs una navegació a l'estima.

El GPS està composat per 3 segments ben diferenciats (figura 1.2): el segment espacial, el segment de control i finalment el de l'usuari. En el segment espacial hi ha la xarxa de satèl·lits que possibilita transmetre el missatge de navegació als usuaris. Aquests retransmeten informació de la seva posició, estat i el temps de transmissió mitjançant rellotges atòmics amb precisió. En el segment de control es *monitoritza* la salut dels subsistemes. Es determina la posició dels satèl·lits a partir del seu seguiment des de terra i es calibren els rellotges dels satèl·lits. Finalment en el segment d'usuari, format per tots els usuaris (civils i militars) del sistema i els seus receptors, es calcula la distància als satèl·lits gràcies a la diferència entre el temps de transmissió i el temps de recepció del

senyal.

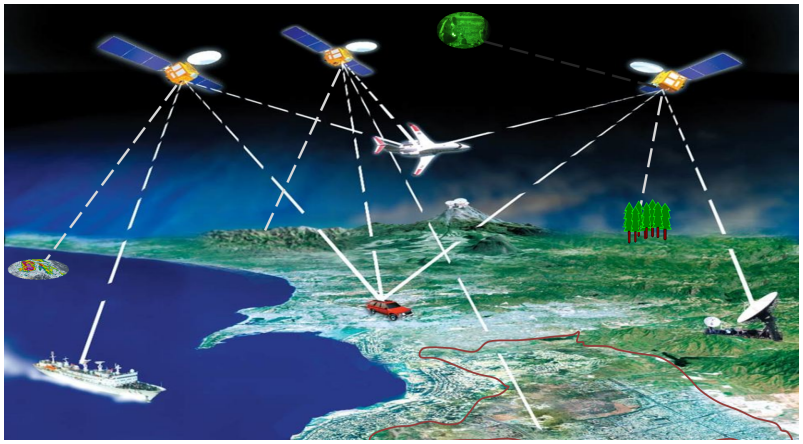

Figura 1.2: Esquemàtic dels diferents segments del GPS

Hi ha moltes variables dins d'aquest càlcul com la geometria que es té dels satèl·lits i la meteorologia, entre molts d'altres. Per tant, el GPS és un sistema molt complex, que en determinades situacions no dóna suficient precisió (acostuma a ser entre metres i desenes de metres), o bé no arriba el senyal correctament. Per tant és necessari contemplar altres opcions en el cas de posicionaments locals que proporcionin una major precisió.

## 1.2. Sistemes de posicionament Locals

En els últims 15 anys s'han desenvolupat i millorat moltes tecnologies ben diferents entre elles, per a la creació de sistemes de posicionament locals [1]. És remarcable la riquesa de possibilitats i opcions que s'han anat trobant aquests darrers anys com a possibles solucions per tal de solucionar el problema del posicionament [10]. Cal esmentar que l'ús dels sistemes de posicionament local *indoor* no està estès per la manca de disponibilitat i precisió que té, comparable amb la del GPS. Entre les possibles solucions que s'han trobat cal destacar la visió per computador mitjançant càmeres, tecnologies de radio freqüència, so i ultrasons, camps magnètics o tecnologies làser. Abans però de començar el desenvolupament d'aquestes tecnologies, cal entendre quin és l'objectiu principal d'aquests sistemes i de quines formes es pot assolir.

S'entén per sistema de posicionament local el sistema de navegació que permet determinar la posició d'un objecte en un àrea o espai local. Altres definicions també inclouen que aquests sistemes són independents de la meteorologia i de l'espai, sempre que s'estigui dins del radi de cobertura del sistema.

Aquests sistemes es poden classificar de forma general en dues branques [2] depenent de qui determina la posició: els sistemes de posicionament local autònoms i els sistemes de posicionament local remot.

En els sistemes de posicionament local autònom, l'objecte a localitzar és capaç de determinar la seva posició autònomament, és a dir, sense necessitat de cap sistema extern. L'interès principal d'aquest sistema és que no hi ha rang de cobertura màxim, o sigui que teòricament el rang és infinit. En la realitat però, no és així ja que el rang està normalment

limitat al rang de l'estació de terra. Aquest sistemes requereixen d'un disseny previ en la construcció de l'objecte i per tant suposen més temps i un cost afegit en la fase de disseny.

En canvi, en els sistemes de posicionament remot, és un agent extern el que s'encarrega de determinar la posició de l'objecte. Aquest fet fa que aquest sistemes siguin més flexibles i adaptables que no pas els autònoms ja que no depenen de l'objecte que s'utilitzi en termes generals. Aquesta independència també implica una reducció en el cost i el temps de disseny de l'objecte però esdevenen uns costs en el disseny de l'agent extern. Aquests sistemes es poden classificar en actius i passius. Els actius són els sistemes on l'objecte té incorporat algun sensor o element que envia informació per tal que l'agent pugui determinar la posició. En els passius és l'agent el que determina la posició de l'objecte sense cap tipus de suport per part de l'objecte. Les diferències entre l'actiu i el passiu són subtils però de forma general es pot afirmar que els sistemes actius suposen una modificació en l'actuació normal de l'objecte, és a dir, alterar el seu comportament per a què enviï una certa informació específica a l'agent.

La classificació general dels sistemes de posicionament es pot apreciar a la figura 1.3

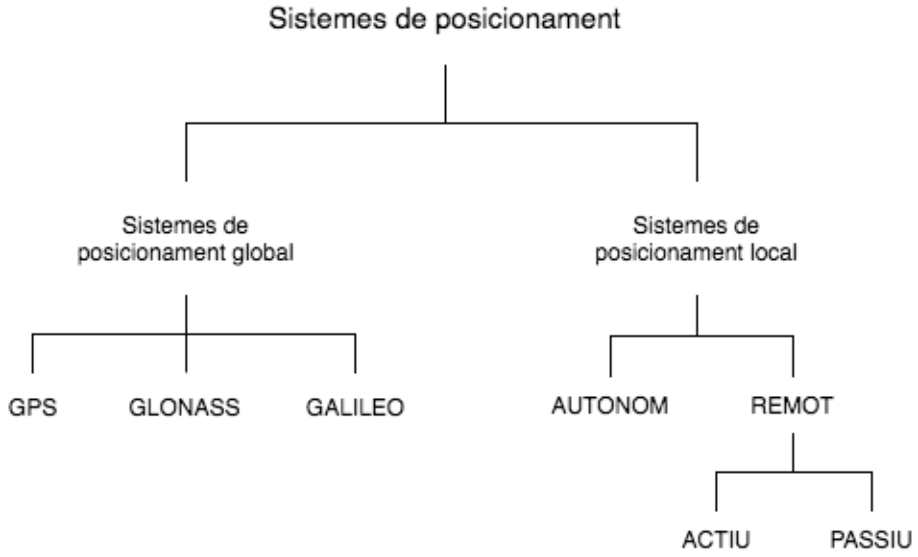


Figura 1.3: Classificació dels sistemes de posicionament

Observant les diferències, els avantatges i els inconvenients dels sistemes autònoms i remots, s'ha decidit continuar amb un sistema remot ja que s'ha prioritzat l'adaptabilitat del sistema i perquè els sistemes autònoms suposen una dificultat afegida.

Hi ha moltes investigacions avui en dia fent recerca d'aquest tipus de sistemes. De forma general aquests es poden classificar segons la tecnologia emprada en: radiofreqüència, visió per computador, magnetisme i làser.

### 1.2.1. Comparativa de sistemes de posicionament local

Tal i com s'ha comentat a l'apartat anterior, actualment existeixen diferents tipus de sistemes de posicionament local [9] o com a mínim molts d'aquests estan en procés d'investigació i proves. D'aquesta manera, abans de triar un d'aquests sistemes basats en

diferents principis, cal una comparativa amb els avantatges i desavantatges de cada un per veure quin d'ells serà el més adequat per usar segons l'objectiu de l'estudi.

Així doncs, les tecnologies més desenvolupades en aquest àmbit són la radiofreqüència, la visió per computador, el magnetisme i el làser. Cada una d'aquestes té els seus propis principis i bases i a simple vista es podria afirmar que, segurament, totes les anomenades aquí podrien servir com a tecnologia per al propòsit del treball. Per això és necessari fer un estudi i una cerca més profunda per acabar escollint l'opció més adient.

Després de buscar diferents medis com poden ser articles o investigacions dutes a terme per diferents universitats d'arreu del món, les descripcions dels diferents mètodes amb les seves característiques i les conclusions pertinents es presenten a continuació.

#### *1.2.1.1. Posicionament basat en radiofreqüències*

El sistemes de posicionament basat en radiofreqüència tenen molta relació amb el sistemes de posicionament globals [6]. Aquests es basen en la retransmissió de missatges via radio per a després poder calcular la posició exacta. Principalment destaquen pel seu cost relativament reduït d'aproximadament desenes de euros, per l'ús de tecnologies sense fils, l'adaptabilitat en els drons ja que gairebé tots disposen de diferents tipus d'antenes i també per usar tecnologies més que provades [7] com podria ser Wi-Fi, BLE, etc. D'altra banda també cal destacar que necessiten menys infraestructures degut principalment a l'ús de tecnologies sense fils.

Per contra, aquests sistemes es veuen limitats per la disponibilitat de les bandes de radiofreqüència i per les possibles interferències degut als obstacles, a l'efecte multicamí, ecos... A més a més, la implementació d'aquests sistemes sol ser complicada i feta per experts de la telemàtica i sovint donen molt poca precisió. L'error de precisió pot arribar a variar molt segons el sistema RF que s'utilitzi, entre desenes de centímetres en els millors casos fins arribar a desenes de metres. Dintre de les opcions, la més precisa és la que usa UWB (de l'anglès *Ultra Wide-Band*).

Així doncs, com s'ha comentat, dintre aquest posicionament basat en radiofreqüència hi ha diversos sistemes desenvolupats amb les seves pròpies característiques:

- UWB (*Ultra Wide-Band*): Una de les principals característiques que fan aquest sistema tant interessant és que la tecnologia UWB pot arribar a assolir un alt nivell de precisió requerit per la majoria d'aplicacions gràcies al temps de resolució tant bo que té. Aquesta qualitat permet un alt nivell de precisió en el mesurament de la distància, de l'ordre de centímetres. A més, proporciona molta robustesa enfront dels fenòmens de trajectes múltiples.

- Wi-Fi: Els beneficis de l'ús de senyals Wi-Fi per al posicionament en interiors provenen principalment de dos punts, per una banda destacar que la inversió en una infraestructura addicional per al posicionament en interiors és innecessària, i per l'altre comentar que el nivell de precisió assolible amb senyals Wi-Fi és més que acceptable per a desenvolupar diversos serveis basats en la localització en entorns rics en senyal Wi-Fi. L'error en la precisió arriba a ser, en els pitjors casos, de l'ordre de pocs metres.

- Bluetooth: És una de les tecnologies que s'ha convertit en un estàndard de la


Figura 1.4: Símbol de la tecnologia Wi-Fi

indústria, fet que ha provocat que estigui disponible a la majoria dels dispositius d'avui dia. En aquesta tecnologia la base són unes balises anomenades BLE (o *iBeacons*) que es caracteritzen per ser barates, petites, tenir una bateria de llarga durada i no requerir una font d'energia externa. El dispositiu detecta el senyal de la balisa i pot calcular aproximadament la distància a la balisa i per tant, estimar la localització. El problema principal és que l'error de precisió, de fins a desenes de metres, és el més elevat dels tres casos presentats en aquesta secció.


Figura 1.5: Símbol de la tecnologia Bluetooth

#### *1.2.1.2. Posicionament basat en magnetisme*

D'altra banda, també existeixen els sistemes de posicionament basats en el magnetisme [5], que aprofiten el nivell de camp magnètic detectat pels sensors per poder determinar i triangular la seva posició. Gràcies a les estructures metàl·liques dels edificis aquests sistemes aposten pel magnetisme com a principal mètode de posicionament [4].

El funcionament d'aquest sistema es basa en "l'empremta magnètica digital" única que té cada edifici o estructura. Aquesta empremta està basada en la manera en com els materials usats en la construcció d'aquest edifici afecten al camp magnètic permanent generat per la Terra. Gràcies a què aquestes distorsions es poden assignar acuradament, es poden situar amb precisió, un o dos metres, els usuaris de *smartphones* o d'altres sistemes compatibles dintre de l'edificació en qüestió, com s'aprecia a la figura 1.6

Cal apuntar que són sistemes que requereixen uns costs mínims, òbviament depèn de la precisió amb la que es pretengui obtenir els resultats però en general no són cars. A més, són sistemes amb poca infraestructura ja que tota la terra compta amb camp magnètic, fet que es reflecteix positivament en l'estalvi de costos. Els principals desavantatges són

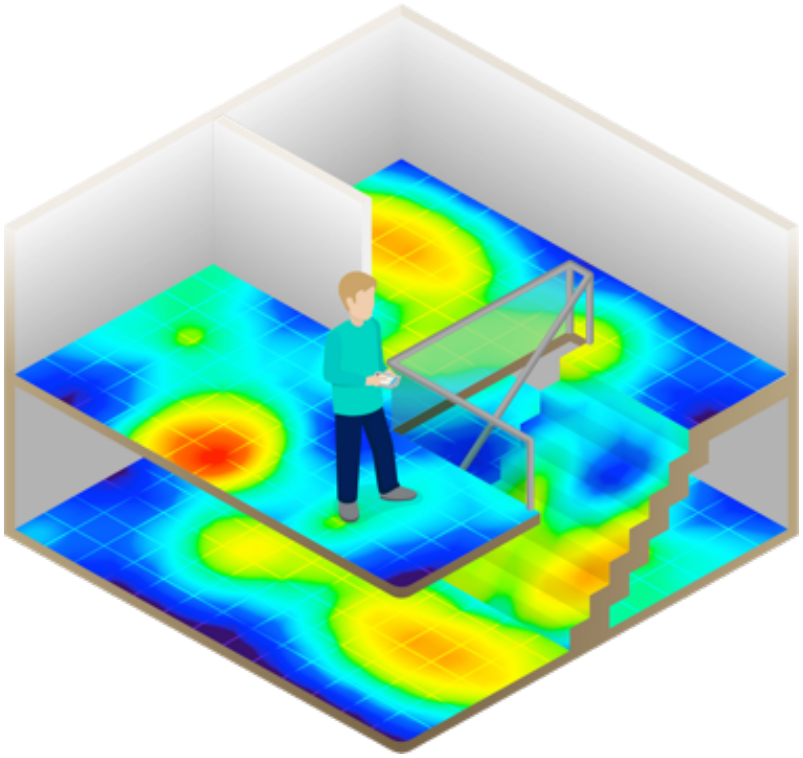

Figura 1.6: Exemple dels sistemes magnètics

la poca integritat que té, les interferències causades per certs objectes metàl·lics com poden ser ascensors en moviment i d'altres, la necessitat de bons sensors magnètics per prendre les mesures i també la irregularitat del camp magnètic i les conseqüents desviacions magnètiques.

#### *1.2.1.3. Posicionament basat en làser*

Una altra de les opcions és el posicionament mitjançant làser que es basa en sensors i mesuradors de distància mitjançant làser [8]. La idea és que, detectant les distàncies respecte diferents objectes, es pot triangular la posició. Aquests sistemes poden arribar a ser molt precisos, d'ordres més petits que milímetres, i ràpids. Les tecnologies làser estan més que desenvolupades i al mercat hi ha un gran nombre de solucions. També aporten una gran robustesa enfront de les condicions externes.

Els principals problemes d'aquest mètodes són bàsicament econòmics, ja què depenent de la precisió que es pretengui obtenir, els sensors làser poden arribar a tenir un preu força elevat (des de centenars fins milers d'euros). A més a més, els làser tenen una alta direccionalitat i per tant, per poder posicionar l'objecte, el sensor ha d'apuntar de forma perfectament correcte. Això suposa la implementació de *gimbals* i estabilitzadors per poder mesurar la distància exacta. També cal tenir en compte que normalment, a part del làser en sí, cal uns certs objectes per complementar el sistema.

En el cas del posicionament de forma autònoma, aquestes característiques suposen augmentar la càrrega de treball del dron i en el cas de posicionament remot suposen tenir uns aparells de *tracking* molt precisos i per tant molt cars.

#### *1.2.1.4. Posicionament basat en visió per computador*

La visió per computador és una opció molt viable per al posicionament. El cost associat a aquestes tecnologies no és tan elevat, ja que només requereix de càmeres. En general aquest tipus de tecnologia no augmenta més d'un màxim d'un centenar d'euros aproximadament i és força més accessible que als usuaris que els altres sistemes descrits prèviament. De forma general, la implementació de la visió per computador no és tan complexa ja que existeixen infinitat de llibreries de codi obert disponibles per a tots els interessats. La visió per computador té l'avantatge de que és molt adaptable a qualsevol tecnologia dron. Això és degut a què gairebé tots els drons actuals disposen de càmeres. Una implementació de forma remota fa que es pugui desenvolupar usant diverses plataformes.

Per contra, aquests sistemes per funcionar en temps real o quasi, requereixen d'una potència de computació considerable (processar entre 10 i 60 imatges per segon) així com de velocitats de transmissió elevades (de l'ordre de milisegons). També és necessari desenvolupar un sistema que permeti identificar els objectes desitjats, ja sigui de forma autònoma o remota. Una vegada descrits tots els sistemes de posicionament local, s'ha construït la taula 1.1 on es poden apreciar les principals característiques de cada sistema de forma comparativa.

Taula 1.1: Comparativa de mètodes de posicionament local

| **Mètode de posicionament** | **Precisió** | **Preu** | **Implementació** |
|---|---|---|---|
| Radio-freqüència | 1-90 m | € | Mitja |
| Magnetisme | $<$ 2 m | €€ | Complexa |
| Laser | 0.001-1 m | €€€ | Complexa |
| Visió per computador | 0.01-1 m | € | Fàcil |

A continuació mostrem uns quants exemples de diferents mètodes o sistemes desenvolupats o en procés de desenvolupament per diferents departaments o universitats d'arreu del món. Com es pot observar la varietat és extensa i és un camp encara amb moltes possibilitats de millora.

- **Wireless Local Positioning**[29]: Estudi fet per una universitat alemana en la que es pretén trobar el posicionament d'un objecte mitjançant el radar primari i secundari i les característiques del *angle of arrival* del senyal.
- **Indoor Positioning System Based on UWB and Inertial Navigation**[30]: Estudi fet per una universitat xinesa que ens permet integrar la tecnologia UWB (*Ultra Wideband*) amb la navegació inercial, i presenta el disseny del sistema de posicionament en interiors híbrid que inclou el servidor, *gateway* i nodes de sensors. Es proposa un model de matriu de regressió lineal per modificar el recorregut de navegació inercial. S'usen mètodes específics per determinar equacions que relacionen les trajectòries de UWB a les trajectòries de navegació inercial.
- **Local Positioning System Based On VHF Band**[31]: Aquest és un treball fet pel departament de recerca de Daejon, Korea. Es proposa un sistema de posicionament local que pot proporcionar l'estimació precisa de la posició utilitzant la banda

de VHF. El sistema proposat té una precisió molt més gran que el sistema GPS utilitzant un gran ample de banda, i una baixa freqüència de RF. Mentre que el GPS utilitza informació sobre el temps per calcular el temps d'arribada entre el transmissor i el receptor, aquest sistema de posicionament local només utilitza informació sobre la diferència de temps d'arribada entre els senyals transmesos.

### 1.2.2. Elecció sistema de posicionament local

Com es pot concloure amb aquests exemples i tenint el compte el que s'ha descrit prèviament, s'observa com la gamma de diferents tecnologies usades per el mateix propòsit és certament elevada. Cal destacar que actualment s'està als inicis d'aquests estudis, com a mínim en l'àmbit civil. Això fa que encara sigui més sorprenent la feina o estudis ja realitzats. Aquest fet fa pensar que en un futur proper, l'atenció que rebrà aquest camp d'estudi anirà en augment i serà molt significativa.

A partir dels avantatges i inconvenients de cada sistema s'ha decidit utilitzar la visió per computador pel cost reduït, la facilitat d'adaptar el sistema a diferents drons, la reduïda complexitat que suposa el sistema i per totes les diferents possibilitats que pot arribar a oferir. Es desenvoluparà doncs aquest sistema, tant des del punt de vista actiu com el passiu.

Els materials necessaris seran:

- Un o varis drons on provar el sistema.
- Una o més d'una càmera.
- Un ordinador on realitzar les proves.
- Una llibreria de visió per computador.
- Un espai o sala suficientment gran com per poder realitzar les proves.
- Un *joystick* per realitzar els primers tests.

# CAPÍTOL 2. VEHICLES AERIS NO TRIPULATS

## 2.1. Elecció dels vehicles

Una vegada decidit quin serà l'àmbit d'estudi en concret, i quins seran els objectius a satisfer durant aquest treball de fi de grau, el següent que cal plantejar és saber quin model de dron serà usat per implementar aquest sistema de posicionament local. Hi ha molts tipus de drons diferents tal i com es pot apreciar a la figura 2.1

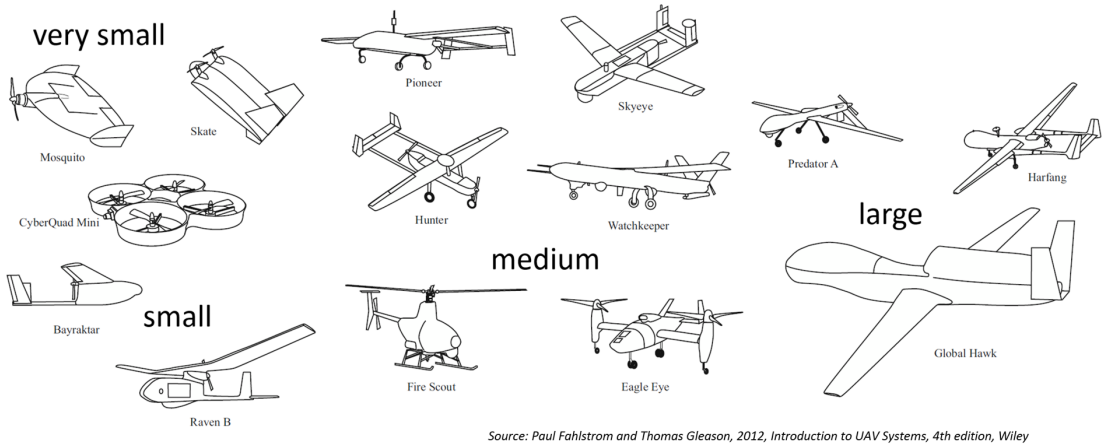


Figura 2.1: Classificació dels UAV segons la seva mida

El dron del tipus quadrotor és l'opció mundialment més estesa dins l'àrea dels drons moderns, i una de les més desenvolupades i avançades en aquest sector. Això és principalment gràcies a les diferents càrregues de treball que pot suportar i a la diversitat de funcions que pot dur a terme. Aquest creixent interès per aquest model de dron en particular prové dels avantatges destacables d'aquestes plataformes voladores, conegudes com a vehicles aeris no tripulats (UAV de les sigles en anglès), en termes de seguretat i eficiència (parlant sempre en termes de mides de dron petites).

D'altra banda, els avenços en la investigació han portat al quadrotor a un nivell d'utilitat que fa que sigui cada vegada més atractiu per a aplicacions comercials com la topografia o el lliurament de productes, entre d'altres coses, perquè té la capacitat de realitzar enlairaments vertical o gràcies a la seva gran capacitat de maniobra. Tot i els atractius que pugui presentar a primera vista, també és cert que aquest tipus de drons encara presenten algunes deficiències que necessàriament hauran de ser arreglades en un futur pròxim per tal de poder arribar a l'excel·lència i ser realment una opció viable per a les aplicacions del món real. Per exemple, els temps de vol, generalment d'ordre de minuts, són encara massa curts per permetre que els quadrotors siguin vehicles d'entrega de productes eficients. D'altra banda els quadrotors tenen una baixa fiabilitat, cosa que pot portar a grans decepcions i fracassos. Aquests i d'altres riscos poden ser mitigats en part augmentant la robustesa i l'eficiència amb què es poden detectar i evitar obstacles.

Així doncs, dos models de drons han estat escollits per aquest estudi. D'una banda el Crazyflie 2.0 de la companyia Bitcraze que servirà per a l'estudi dels sistemes passius i

per altra banda el AR Drone 2 de la companyia Parrot, per als sistemes actius. Ambdós drons se'ns han proporcionats per la universitat.

## 2.2. Crazyflie 2.0

Per realitzar aquest estudi i poder fer tests es va decidir primerament usar el dron Crazyflie 2.0, un tipus de quadrotor nano. El desenvolupament del Crazyflie va començar l'any 2009 com a projecte de desenvolupament per tres treballadors de la consultora Epsiolon AB. Al 2010, i després de presentar un vídeo en societat sobre un primer prototip d'aquest dron que va crear una expectació molt elevada, els tres creadors del Crazyflie van decidir crear Bitcraze AB, que poc després es convertiria en Bitcraze.se per poder finançar el desenvolupament i fabricació del quadrotor. Primer es va llençar al públic una versió 1.0 a la vegada que en la seva plataforma i el seu desenvolupament de codi obert, però pocs mesos més tard va sortir la versió Crazyflie 2.0 amb unes característiques molt millors que les de la versió prèvia [11]

El Crazyflie 2.0 (figura 2.2) és un dels quadcòpter actualment més petits del mercat, fet que juntament amb la seguretat que proporciona durant el vol, sigui ideal per medis tancats i amb obstacles o persones. Entra dins de la categoria de mini drons pel seu tamany reduït però té unes *performances* considerablement bones.

Figura 2.2: Imatge del Crazyflie 2.0

Consta d'unes mesures de 9.2 cm d'hèlix a hèlix i té un pes de 27 grams incorporant senyals visuals. Tenir un quadcópter nano que pesa solament 27 g té molts avantatges. La mida el fa ideal per a volar dins d'un laboratori, oficina, o una sala d'estar amb la possibilitat

de no destrossar gran part de l'interior. Tot i que les hèlixs giren a altes revolucions, són suaus i el parell en els motors és molt baix en comparació amb un *brushless* motor (ECMs). El Crazyflie 2.0 pot arribar a ser bastant ràpid si se'l permet ser-ho, però cal mencionar que en el cas de xocar amb algun objecte, el fet de pesar només 27 g l'energia cinètica involucrada en l'accident seria bastant baixa. Durant un impacte dur, el sistema està dissenyat per trencar-se pel component més econòmic, els suports del motor, que estan disponibles com a peces de recanvi. També comentar que té una autonomia de vol d'aproximadament 7 minuts a ple rendiment.

Les seves petites dimensions el doten d'una gran varietat de possibles usos com a plataforma de desenvolupament i noves recerques. Un dels primers inconvenients que suposa el seu tamany petit, i que es poden veure a simple vista, és la seva limitació a l'hora de parlar sobre la càrrega de treball que pot suportar, fet que incrementa la necessitat de treballar amb controladors que suposin un temps de resposta (latència) baix. El fet d'incorporar diversos sensors i usar-los a ple rendiment comporta un altre desafiament de cara a l'ús total del Crazyflie 2.0.

Un dels aspectes més destacables d'aquest dron és que tot el *software*/codi desenvolupat per Bitcraze és obert i en constant actualització [13][14]. Això permet un control total sobre *firmware* del quadrotor, el *firmware* que s'executa a la ràdio, així com la biblioteca de client que s'executa a l'estació base. Hi ha una comunitat gran de desenvolupadors que contínuament estan fent millores en el codi arrel ja que Bitcraze allotja la totalitat de la base de codi en Github, un sistema de compartició publica de codi. És principalment per aquest motiu, el fet de tenir una possibilitat de total manipulació de codi, pel que s'ha decidit escollir aquest dron per al desenvolupament de l'estudi.

### 2.2.1. Hardware del Crazyflie

#### *2.2.1.1. Microcontroladors del Crazyflie*

El Crazyflie escollit per dur a terme les proves del nostre estudi, el model 2.0, consta de dos microcontroladors. El principal, que s'utilitza amb l'objectiu d'executar l'aplicació principal, és un processador encastat ARMCortex-M4 (STM32F405). Aquest processador té una arquitectura de 32 bits i pot arribar a funcionar a 168 MHz. El Crazyflie 2.0 també està composat per un segon microcontrolador per tal de poder gestionar la potència i la radio. Aquest segon microcontrolador és un microprocessador ARM Cortex-M0 (nRF51822) que funciona a 32 MHz. Aquest segon microcontrolador té la característica de ser compatible amb bluetooth.

Especificacions:

- STM32F405 main application MCU (Cortex-M4, 168 MHz, 192 kb SRAM, 1 Mb flash)
- nRF51822 radio and power management MCU (Cortex-M0, 32 Mhz, 16 kb SRAM, 128 kb flash)
- uUSB connector
- On-board LiPo charger with 100 mA, 500 mA and 980 mA modes available

- Full speed USB device interface
- Partial USB OTG capability (USB OTG present but no 5 V output)

#### *2.2.1.2. Subsistemes inercials del Crazyflie*

El quadcopter Crazyflie 2.0 està equipat amb una IMU, l'MPU-9250. Aquesta IMU conté un giroscopi de 3 eixos així com un acceleròmetre també de 3 eixos i un magnetòmetre de 3 eixos, fet que proporciona 9 graus de llibertat. D'altra banda aquesta IMU compta d'un baròmetre per mesurar pressions atmosfèriques i un sensor de temperatura. Aquests elements, a l'estar implementats dins del mateix component, evitaran l'ús de més cablejat i proporcionaran dades útils per realitzar el control i recollida de dades.

Especificacions:

- 3 axis gyro (MPU-9250)
- 3 axis accelerometer (MPU-9250)
- 3 axis magnetometer (MPU-9250)
- high precision pressure sensor (LPS25H)

#### *2.2.1.3. Bateries del Crazyflie*

La bateria utilitzada pel Crazyflie 2.0 (figura 2.3) es tracta del tipus més popular de bateria a la indústria de R/C i ofereix la millor relació pes-potència. Dient aquestes característiques només es pot tractar d'una bateria d'una sola cèl·lula de polímer de liti (Li-Po). Es subministra 3.7 volts i té una capacitat de 240 mAh. La bateria també té un mòdul de protecció de circuits (PCM) unit que evita que l'usuari pateixi sobrecàrrega de la bateria o un curtcircuit, ja què un dels perills que ens podem trobar usant aquesta bateria és que l'exposició d'aquesta a la llum solar directa pot causar un sobreescalfament, el que podria donar lloc a la invalidació o un incendi. Així doncs, i a més a més, la bateria ha de ser carregada amb el carregador adequat, d'acord amb la instrucció de l'operació correcta.

És fàcil extreure la bateria de la quadrotor. La seva velocitat de descàrrega és de 15C, que en teoria hauria de proporcionar 4 minuts de vol continu.

Especificacions:

- Operating Temperature: 0-35 $^{o}$C
- Connector: Molex 51005-2P (Pin 1 positive)
- Capacity: 240 mAh
- Nominal voltage: 3.7 V (1 Cell)
- Discharge: 15 C
- Charge: 2 C
- Weight: 7.1 g

- Size (WxHxD): 20x7x30 mm

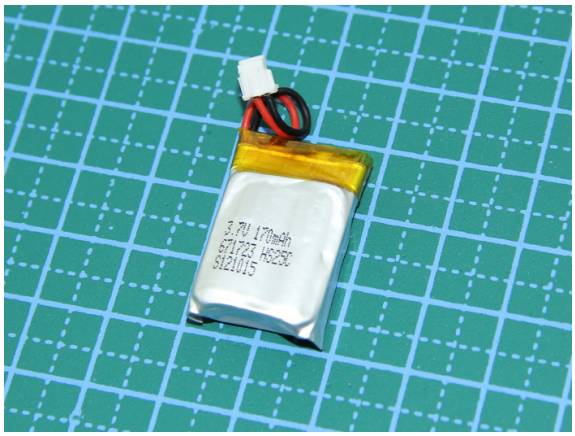

Figura 2.3: Bateria del Crazyflie 2.0

#### *2.2.1.4. Motors del Crazyflie*

Quan es tracta dels motors, cal comentar que el Crazyflie 2.0 amb el qual es treballarà té quatre motors de corrent continu raspallat (*brushed*). Els motors (figura 2.4) són sense nucli fet que provoca en teoria una acceleració més ràpida. Poden produir 12000 rpm per volt, amb una tensió nominal de 4.2 V.

Una de les característiques més interessants sobre els motors és que la majoria de quadrotors utilitzen motors sense escombretes (EC motos), que utilitzen un circuit electrònic per regular amb precisió la seva velocitat respecte al senyal d'entrada. En canvi en el nostre cas, els motors 2.0 Crazyflie són amb escombretes, i són alimentats per una font d'energia no regulada.

- Diameter: 7.0 mm
- Length: 16.0 mm
- Shaft length: 3.5 mm
- Shaft diameter: 0.8 mm
- Weight: 2.7 g
- Wire length: 32.0 mm
- Kv: 14 000 rpm/V
- Rated voltage: 4.2 V
- Rated current: 1000 mA

Figura 2.4: Motor del Crazyflie 2.0

#### *2.2.1.5. Sistema de propulsió del Crazyflie*

El dron consta de 4 hèlix convencionals de plàstic d'una mida de 45 mm (figura 2.5). Un dels problemes més recurrents durant els experiments és la facilitat amb què aquestes hèlix es dobleguen després d'una col·lisió i, per tant, cal tornar-les a calibrar totalment o, en cas d'estar molt danyada, cal canviar les hèlix per altres de noves per tal de poder continuar amb el vol en condicions òptimes. Per calibrar les hèlix una altre vegada s'han de seguir aquests tres passos:

1. Primer de tot mirar si l'hèlix està descalibrada. Per fer això cal desmuntar l'hèlix i fer passar una agulla pel seu centre (pel seu eix). Si al posar l'hèlix en posició horitzontal aquesta no es manté en aquesta posició sinó que s'inclina cap a alguna banda, podem afirmar que aquesta hèlix esta descalibrada.

2. Per tal de calibrar una hèlix senzillament és necessari enganxar un tros petit de cinta adhesiva a la part posterior de l'hèlix. La quantitat o el grossor d'aquest tros dependrà del grau de descalibració que sofreixi l'hèlix.

3. Per últim només cal tornar a realitzar la prova amb l'agulla en el centre de l'eix. En aquest cas, i si ho hem fet bé, l'hèlix es mantindrà en posició horitzontal sense moure's.

#### *2.2.1.6. Ràdio del Crazyflie*

La millor manera de poder guiar al dron és mitjançant la radio (figura 2.6). El Crazyflie està equipat amb un Nordic Semiconductor nRF51822 que s'encarerga de la comunicació ràdio. Per fer-ho, l'opció més fàcil és usar el Crazyradio. El Crazyradio és bàsicament un USB amb antena que integra diversos dispositius que permeten la comunicacio amb el dron. Els xips es poden comunicar entre si a través de la ISMband 2,4 GHz. Tots dos xips poden ser reprogramats, tot i que en el cas del del Crazyflie és necessari un connector JTAG. Una altra de les característiques és que les ràdios es poden executar fins a una velocitat de 2 Mbs.

- 20 dBm radio amplifier tested to 1 km range LOS with Crazyradio PA

Figura 2.5: Hèlix del Crazyflie 2.0

- Bluetooth Low Energy support with iOS and Android clients available (tested on iOS 7.1+ and Android 4.4+)
- Radio backwards compatible with original Crazyflie Nano and Crazyradio

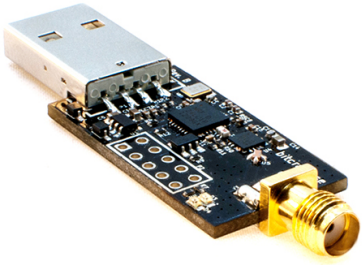

Figura 2.6: Ràdio del Crazyflie 2.0

Per últim, una vegada comentades les parts més importants que composen el Crazyflie també cal esmentar les estructures auxiliars que permeten que tots aquests dispositius funcionin i estiguin perfectament integrats dintre el tot que suposa el dron.

#### *2.2.1.7. Suports del motor del Crazyflie*

Aquestes peces són bàsicament uns suports (figura 2.7) que permeten connectar i facilitar la connexió entre la part principal del dron, la placa base, amb els motors i les hèlix.

#### *2.2.1.8. Targeta d'expansió del Crazyflie*

Aquest producte està dissenyat per ser utilitzat juntament amb el Crazyflie quadcopter 2.0 nano i usa el seu sistema de targeta d'expansió (figura 2.8). Senzillament és una placa que permet una millor integració de tot el sistema i, a més, també facilita la seva protecció.

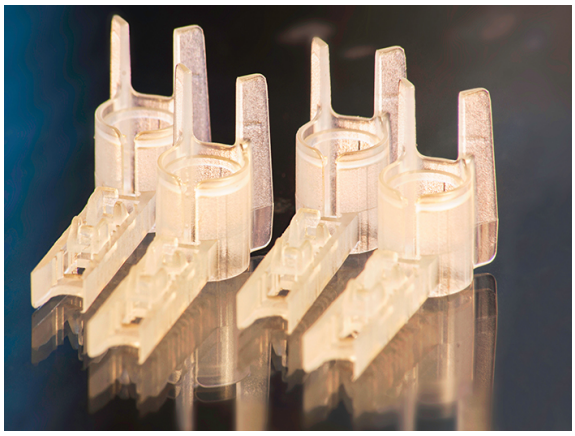

Figura 2.7: Suports del Crazyflie 2.0

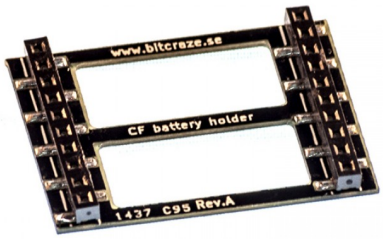


Figura 2.8: Expansió del Crazyflie 2.0

### 2.2.2. Software del Crazyflie

Com ja s'ha comentat amb anterioritat, un dels avantatges del Crazyflie, i motiu pel qual es va escollir com a opció a desenvolupar, és que tot el *software*/codi de Bitcraze és obert i en constant actualització. En el cas del Crazyflie 2.0 el seu *firmware* està basat en el FreeRTOS que no és res més que un sistema operatiu que es caracteritza per tenir un codi obert a temps real. Aquest *firmware* distribuit i usat pel Crazyflie es basa en executar una gran gamma diferent d'ordres o serveis anomenats tasques (*tasks* en anglès). Les tasques enviades al dron com a ordres s'anomenen *commands* i s'envien mitjançant el Crazy RealTime Protocol (CRTP). En la capçalera d'aquest paquet del CRTP hi trobem tres camps diferents. Primer el port, que especifica a quina aplicació va el paquet. A continuació el segon camp és l'enllaç que actualment està sense usar. Per últim trobem el camp del canal que ens indica la funcionalitat.

Com que es tracta d'un dron *open-source*, tot el codi *software* es pot trobar a la pàgina de distribució [12]. Hi ha molts projectes però en podríem destacar:

- Crazyflie-clients-Python: aplicació feta en Python que permet controlar el Crazyflie gràcies a un *joystick*, amb possibilitat de canviar molts paràmetres i visualitzar la telemetria.
- Crazyflie-lib-Python: llibreria de Python que permet enviar comandes a través de la radio directament al Crazyflie. L'aplicació utilitza aquesta llibreria per elaborar els paquets necessaris i enviar-los mitjançant la ràdio.
- Crazyflie-android-client: aplicació per *smartphones* per poder controlar el dron de forma senzilla.
- Crazyflie-vm: màquina virtual que conté totes les llibreries i projectes així com totes les configuracions necessàries per poder començar el desenvolupament del Crazyflie des de zero.

La corba d'aprenentatge d'aquest tipus de projecte és molt elevada i més encara si no es coneix el principal llenguatge de programació que és Python[15].

### 2.2.3. Desenvolupament i experiments del control del Crazyflie

Per poder tenir un control del Crazyflie s'han realitzat diferents experiments. De forma seqüencial s'ha anat incrementant el control del dron. L'objectiu d'aquesta part de l'estudi és trobar un sistema de navegació que permeti controlar el Crazyflie de forma autònoma.

Es va començar amb experiments amb el client d'android, és a dir, des d'un *smartphone*, després amb el client Python d'ordinador amb un *joystick* com a controlador i finalment de forma autònoma mitjançant les llibreries obertes de Python. En els dos primers mètodes el controlador serà un pilot humà, en canvi en l'últim dels casos serà amb l'ordinador.

Taula 2.1: Resum dels experiements de control del Crazyflie

| | **Android client** | **Python client** | **Python lib** |
|---|---|---|---|
| Número de proves | 15 | 100 | 150 |
| Forma de connexió | Bluetooth | Crazyradio | Crazyradio |
| Controlador | Smartphone | Joystick + Ordinador | Ordinador |
| Autonomia | 8 min | 8 min | 9 min |
| Estabilitat | poca | molta | mitja |
| Rang de connexió | $<$ 5 m | $<$ 10 m | $<$ 10 m |
| Control d'altitud | No | Si | No |
| Complexitat | baixa | mitja | alta |
| Flexibilitat | baixa | mitja | alta |

#### *2.2.3.1. Experiments usant el client d'android*

El primer pas va ser muntar el dron amb totes les seves peces i motors i comprovar que tots els components estiguessin en bon estat i funcionessin. Dins d'aquest manteniment

inicial també es va calibrar les pales del motors tal i com s'ha indicat amb anterioritat a l'apartat del Crazyflie per tenir un vol equilibrat.

Les primeres proves que vàrem realitzar van ésser a través del client d'android (figura 2.9), és a dir, connectant-nos al dron amb els telèfons mòbils mitjançant l'aplicació que es pot obtenir completament gratuïta per Google Play. En començar a fer proves amb el Crazyflie vàrem observar dos aspectes importants: el primer va ser que l'autonomia del dron és reduïda (aproximadament d'uns 7-8 minuts) i d'altra banda que el control era força més complicat del que ens podíem imaginar degut a la poca estabilitat que ofereix el dron.

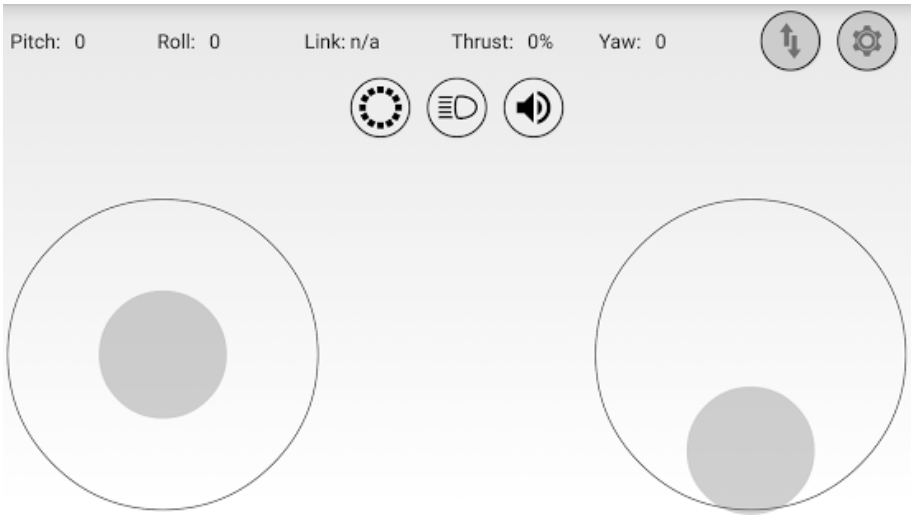


Figura 2.9: Aplicació d'android per controlar el Crazyflie

L'aplicació s'anomena Crazyflie Client i el seu funcionament és molt bàsic. Es compon d'un parell de botons que ens permeten variar principalment el *thrust*, augmentant-lo o disminuint-lo al nostre gust, i les tres rotacions possibles segons els tres eixos: *yall*, *pitch* i *roll*. Aquesta aplicació es compon principalment de dos botons que ens permeten la maniobrabilitat de l'aparell. Segons s'utilitzin els botons en mode vertical o horitzontal aconseguirem modificar els quatre paràmetres com s'aprecia a la figura 2.10

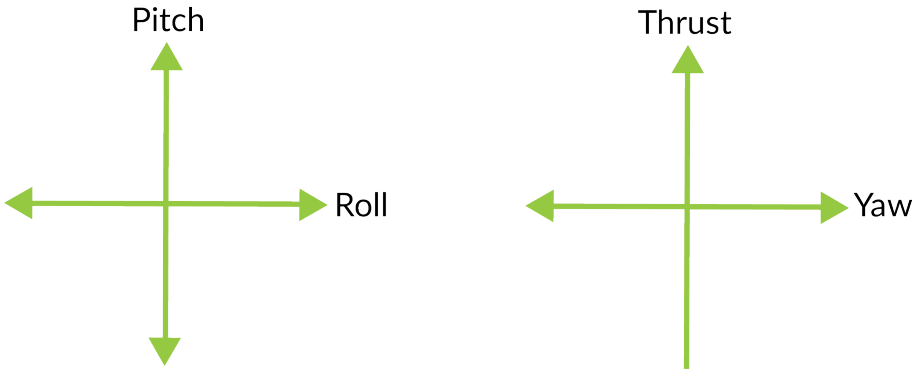


Figura 2.10: Mode 1 del control del Crazyflie amb l'aplicació d'android

En el cas del botó esquerra es podrà variar tant el *pitch* com el *roll*. Òbviament seguint la lògica, el *pitch* es variarà usant el botó en vertical mentre que el *roll* ho farà sempre que desplacem el botó cap a la dreta o l'esquerra. D'altra banda amb l'altre botó el que aconseguirem modificar seran els dos paràmetres restants, el *thrust* i el *yaw*. En aquest cas,

en vertical augmentarem o disminuirem el *thrust* mentre que si el movem horitzontalment aconseguirem una variació en el *yaw*. Una vegada explicat el funcionament de l'aplicació en si també cal comentar que hi ha una opció d'ajustos que et permet adaptar diferents paràmetres com la connectivitat, la sensibilitat i d'altres paràmetres. Així doncs, vam poder modificar certs aspectes dins de l'aplicació mòbil per tal de tenir una navegació més suau cosa que es fa reflectir satisfactòriament en el vol. Tal i com s'ha comentat en un apartat anterior, la connexió amb el Crazyflie es realitza mitjançant l'antena bluetooth que té incorporada. És molt adient doncs que el dron disposi de dues antenes per la comunicació, ja que això facilita molt els problemes que puguis sorgir a l'hora de connectar-se. Un apunt que cal fer és que el radi de cobertura d'aquesta comunicació és aproximadament d'uns 10 metres, per tan no vam tenir problemes per aquesta part per realitzar els experiments dintre d'una sala. Hem pogut observar, òbviament, que no a tothom se li dona igual de bé el control o manipulació d'aquests aparells, per tan es pot afirmar que la capacitat de control del dron es veu molt limitada a les capacitats tècniques del pilot que l'estigui usant.

Al llarg d'aquests experiements també es va observar que el dron era bastant robust a les caigudes des de poca alçada, però en cas d'estar voltant a més de 5 metres i sobtadament perdre la comunicació, certes parts del Crazyflie com els suports dels motors i les hèlixs es podien malmetre. Per tant una de les mesures que vam prendre per realitzar experiments va ser esmorteir els cops del Crazyflie afegint unes defenses al dron i posant material més tou al terra del laboratori com una catifa (figura 2.11)

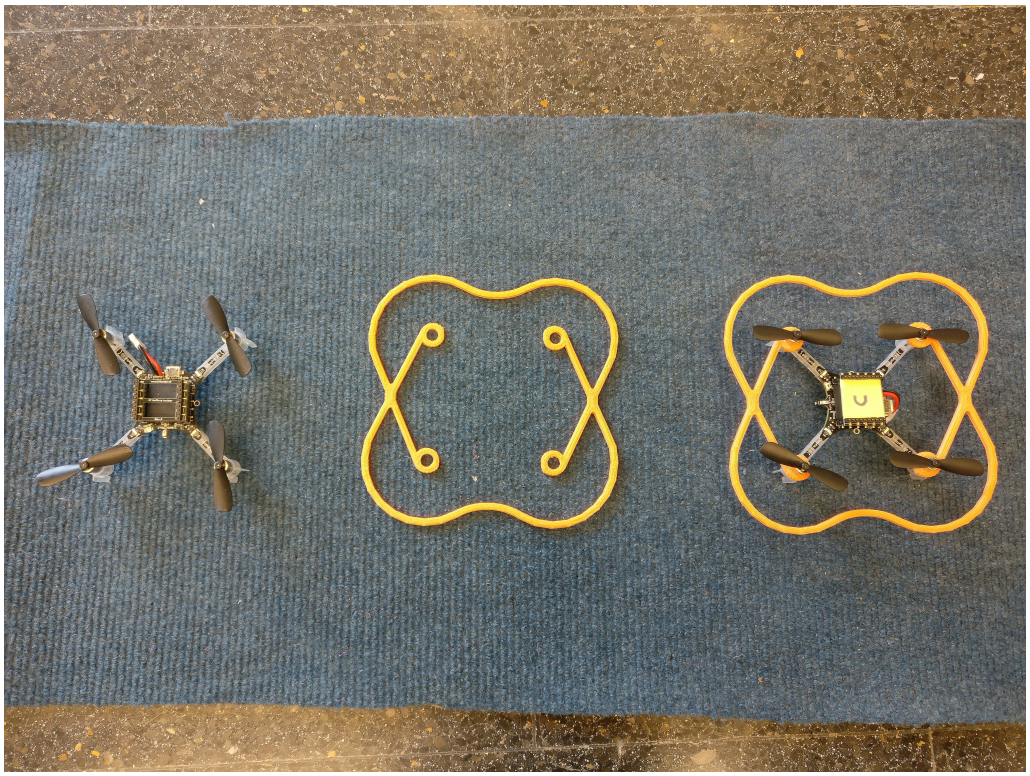

Figura 2.11: Mesures de protecció del Crazyflie

#### *2.2.3.2. Experiments usant el client de Python*

Una vegada provada l'aplicació per mòbil i ja familiaritzats amb el dron amb el que es faran les proves, el segon pas va ser el de realitzar el control del Crazyflie a partir de l'aplicació d'escriptori (figura 2.12) en Python i l'antena radio. Primerament vàrem instal·lar la màquina virtual[16] on estaven ja configurades les llibreries necessàries mitjançant un *link* que ens proporcionava la pròpia companyia Bitcraze a la seva pàgina web (tant en opció torrent com en descàrrega directa). A continuació, l'únic que ens faltava era instal·lar les llibreries i tot el *software* necessari de forma nativa per poder optar a un processament més ràpid, això és degut bàsicament a la necessitat de controlar el dron amb visió per computador. La instal·lació dels diversos mòduls va ser força complicada.

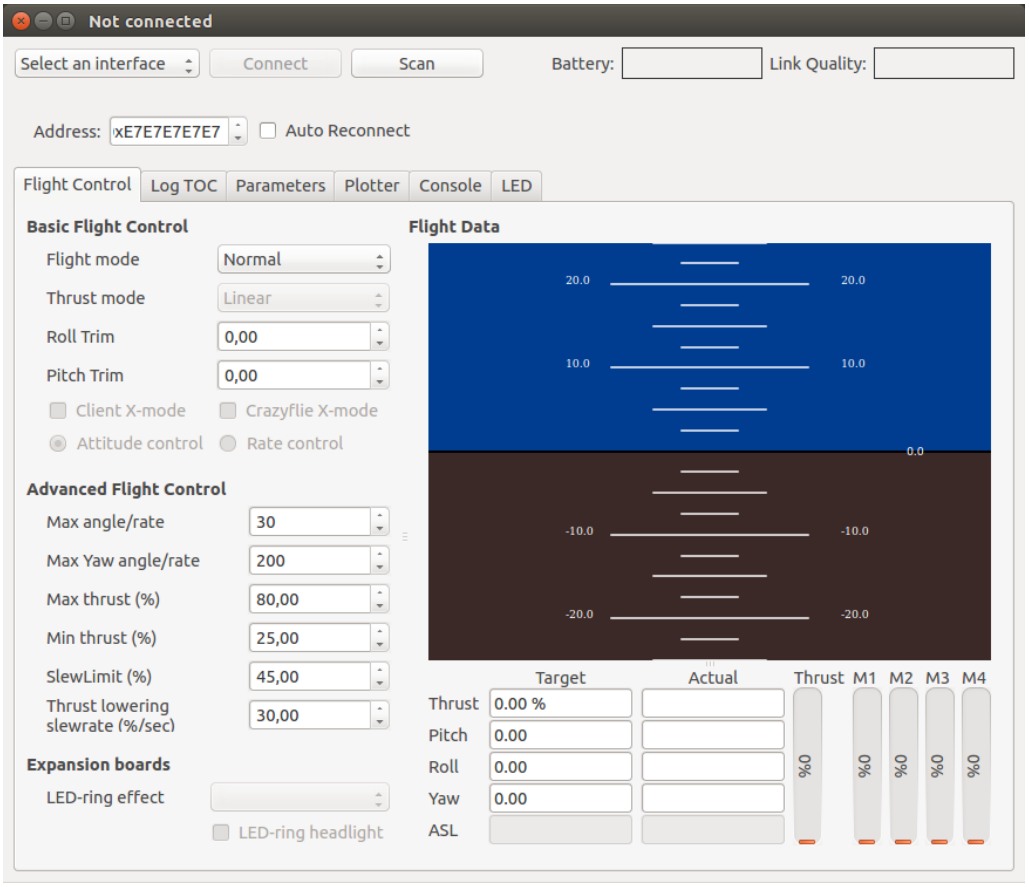


Figura 2.12: Client de Python del Crazyflie

Amb les dues combinacions possibles, tant des de la maquina virtual com de forma nativa, el control va ser més fàcil i accessible, ja que es va configurar el sistema amb una palanca de control (*joystick*) com s'ilustra a la figura 2.13. Per tant, la maniobrabilitat en el vol era superior que des de l'aplicació mòbil. El mateix problema de l'autonomia però, seguia persistint. Un dels avantatges que ofereix aquesta eina és la possibilitat de calibrar el dron i poder modificar paràmetres interns com podrien ser els PIDs, el *trimatge* i l'adreça de comunicació així com observar el nivell de bateries o l'actitud posicional del dron. Cal destacar que abans d'usar el *joystick*, vam voler fer un pas previ entre el mòbil i la palanca de control que va ser senzillament usar el comandament d'una PlayStation 3 (una videoconsola). Aquest està format per dos petits *joysticks* que ens va semblar un bon pas previ mantenint de manera semblant els dos botons que oferia l'aplicació mòbil

però ja usant un comandament extern.

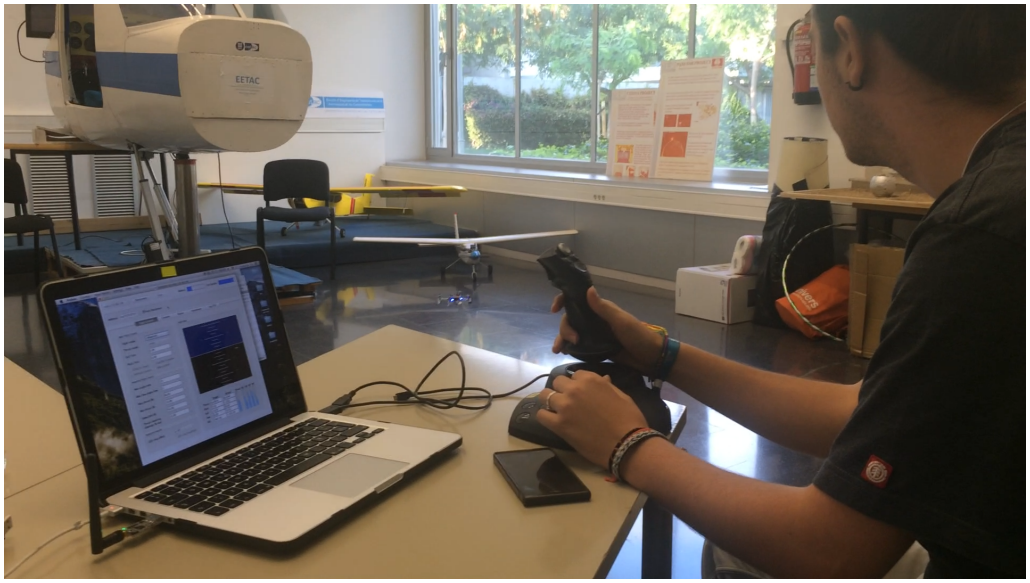

Figura 2.13: Experiments usant el client de Python del Crazyflie

Després de fer un gran número de vols inicials i de tornar a revisar apunts sobre la matèria, al final, modificant certs paràmetres, es va aconseguir tenir un vol relativament estable i controlat. Els valors de *trimatge* es varen enregistrar així com les actuacions (*performances*) del dron: l'empenta necessària per l'enlairament, el mínim de bateria necessari per a poder funcionar, quins canals de radio funcionen millor a les condicions del laboratori, etc.

Els valors configurats es poden apreciar a la taula 2.2:

Taula 2.2: Taula de parametres de calibració

| **Paràmetre** | **Valor** |
|---|---|
| `Flight Mode` | Advanced |
| `Roll Trim` | 3 |
| `Pitch Trim` | -6 |
| `Max angle/rate` | 12 |
| `Max Yaw angle/rate` | 10 |
| `Max thrust (%)` | 80.00 |
| `Min thrust (%)` | 10.00 |
| `Slew Limit (%)` | 55.00 |
| `Thrust lowering slewrate (%/s)` | 5.00 |

Cal destacar però que tant els valor de *roll*, *trim* com *pitch* variaven en funció de quin Crazyflie s'utilitzés donat que en teníem dos per fer les proves pertinents.

Gràcies a les millores de seguretat que vam aplicar, el dron va patir menys col·lisions. El radi de cobertura del dron en aquest cas es manté constant degut sobretot a interferències amb altres xarxes sense fils. Depenent de l'ample de banda escollit i del canal, la comunicació era impossible de connectar amb el dron. Una de les coses que es va poder observar de forma experimental va ser com a 250 kHz i usant el canal de 80 Hz si

que podíem obtenir una bona comunicació amb el dron. També gràcies a aquest client vàrem poder modificar les adreces que venien de fabrica dels drons per poder controlar-los individualment i així poder escollir quin dron seria controlat.

Ambdues formes de connectar-se al dron (des d'android i amb l'aplicació d'escriptori) ens varen servir per aprendre molts coneixements bàsics com per exemple aprendre a volar amb el Crazyflie, les mesures de prevenció que havíem de prendre a l'hora de realitzar els experiments i fins i tot, ens va fer començar a adonar de les dificultats que podríem tenir a l'hora de dur a terme el control.

De tota manera, per poder incorporar el sistema de posicionament i fer un control autònom del dron, ens era necessari poder comunicar-nos amb el dron sense necessitat d'un pilot, és a dir, d'una forma totalment autònoma. Per tant una vegada ja es dominava totalment el Crazyflie des de l'ordinador mitjançant l'aplicació de l'escriptori, es va anar un pas més enllà i es va decidir començar a realitzar experiments amb la llibreria del Crazyflie.

### 2.2.4. Experiments usant la llibrería de Python

Un cop es va considerar que ja s'estava prou familiaritzat amb la màquina virtual, el següent pas ja era el pas definitiu: treballar amb les llibreries del Crazyflie mitjançant Python. Usant aquest tipus de connexió és on tenim la màxima flexibilitat i possible *customització* del control, però també cal afegir que és l'opció més complexa ja que gairebé tot el que es programarà s'ha de fer de zero.

Els primers passos obvis per a poder usar aquesta llibreria és descarregar-se-la i instal·lar-la per poder-la usar com a mòdul de Python. Es tracta d'un procés complex i no pas curt, però seguint certes indicacions i passos al final vam aconseguir-ho.

Per començar s'estableix la connexió amb el Crazyflie i per això es selecciona un dels canals a la freqüència que transmet el Crazyflie.

```
# Initialize the low-level drivers (don't list the debug drivers)
cflib.crtp.init_drivers(enable_debug_driver=False)
# Scan for Crazyflies and use the first one found
print('Scanning interfaces for Crazyflies...')
available = cflib.crtp.scan_interfaces()
print('Crazyflies found:')
for i in available:
    print(i[0])

cf = Crazyflie() # Init the Crazyflie class
cf.open_link(i[0][0]) # Open link connection
```

Un cop establerta la connexió amb el dron hi ha tres tipus diferents de comunicacions possibles que es poden dur a terme amb el Crazyflie. Primerament trobem els *parameters*, a continuació els *logs* i per últim tenim els *commanders*.

- Els `paràmetres` són constants que afecten el control del dron i certs aspectes del seu comportament com podria ser els PID, els límits, els modes, etc.
- Els `logs` són registres de variables que registra el dron amb els seus sensors, com per exemple la temperatura, els acceleròmetres, els giròscops, etc.

- Finalment el `commander` és una funció senzilla que permet enviar una instrucció al dron com per exemple augmentar el *thrust* o girar a l'esquerra.

Utilitzant la llibrería del Crazyflie s'han *testejat* els programes indicats a la taula 2.3 següent, cadascun amb un objectiu específic. Fent millores sobre la base inicial per tal de poder tenir un sistema de control autònom pel Crazyflie.

Taula 2.3: Programes realitzats amb la llibreria de Python

| Nom del programa | Proves | Descripció |
|---|---|---|
| `Basic_ramp` | 10 | encendre els motors |
| `Basic_log` | 20 | rebre informació dels logs |
| `fall_safe` | 40 | mesura de seguretat quan cau |
| `graphic_fall_safe` | 10 | graficar l'acceleració |
| `commander_thread` | 20 | fer un fil apart pel commander |
| `control_thread` | 50 | fer un fil apart pel control |
| `main_control` | 50 | integrar tots els fils |

El primer que vàrem provar van ser els exemples que es troben dins mateix de la llibreria. Aquests són el `Basic_ramp` i el `Basic_log` . El `Basic_ramp` el que fa és establir la connexió amb el Crazyflie, activa els motors i fa una rampa amb el *thrust*, és a dir comença a 0% de potència i arriba fins al 50% de la potència d'empenta. Aquest programa ens serveix per entendre el funcionament dels motors, aprendre a activar-los, com enviar els valors de *thrust* desitjats i com tancar la connexió amb el Crazyflie de forma correcta. Un aspecte que vàrem observar era que si es perdia la connexió amb el dron en ple vol, aquest queia de forma brusca i molt sovint es trencaven els suports dels motors. A continuació podem observar part del codi que genera aquesta rampa amb la següent funció de Python.

```
def _ramp_motors(self):
    thrust_mult = 1
    thrust_step = 500
    thrust = 20000
    pitch = 0
    roll = 0
    yawrate = 0

    # Unlock startup thrust protection
    self._cf.commander.send_setpoint(0, 0, 0, 0)

    while thrust >= 20000:
        self._cf.commander.send_setpoint(roll, pitch, yawrate, thrust)
        time.sleep(0.1)
        if thrust >= 25000:
            thrust_mult = -1
        thrust += thrust_step * thrust_mult
    self._cf.commander.send_setpoint(0, 0, 0, 0)
    # Make sure that the last packet leaves before the link is closed
    # since the message queue is not flushed before closing
    time.sleep(0.1)
    self._cf.close_link()
```

El `Basic_log` en canvi no activa els motors en cap moment. Aquest programa serveix per començar a rebre informació dels *logs* del Crazyflie. Primerament, abans de començar a rebre res, s'han de definir aquests *logs*, quantes variables s'enviaran en cada paquet, quines variables estan disponibles, cada quant temps s'enviaran els *logs* i com activar les respostes a diversos successos, com per exemple quan es rep un *log*, quan es perd la connexió, quan no està ben configurat el *log*, etc. A partir de les proves amb aquest programa vàrem determinar de forma empírica que el màxim nombre de variables que podria contenir un *log* són 3. També que l'interval de temps entre *logs* havia de ser entre 50 i 200 ms per a poder tenir una resposta correcta. Finalment el que es va programar amb aquest exemple és que quan es rebés un *log*, s'imprimís per consola. Vàrem tenir el problema de no poder imprimir tota la informació per pantalla si l'interval de temps era molt petit ja que n'hi havia massa. També es va observar amb aquest programa, que hi havia moltes variables a les quals es podia tenir accés, entre les que es van destacar els estabilitzadors d'actitus, els valors mesurats pels acceleròmetres, els giròscops, el nivell de les bateries, la temperatura...

A continuació podem veure a mode d'exemple part del codi que genera aquest *log* en el moment en que el Crazyflie esta connectat a la radio.

```
def _connected(self, link_uri):
    """ This callback is called form the Crazyflie API when a Crazyflie
    has been connected and the TOCs have been downloaded."""
    print('Connected to %s' % link_uri)

    # The definition of the logconfig can be made before connecting
    self._lg_stab = LogConfig(name='Stabilizer', period_in_ms=10)
    self._lg_stab.add_variable('stabilizer.roll', 'float')
    self._lg_stab.add_variable('stabilizer.pitch', 'float')
    self._lg_stab.add_variable('stabilizer.yaw', 'float')

    # Adding the configuration cannot be done until a Crazyflie is
    # connected, since we need to check that the variables we
    # would like to log are in the TOC.
    try:
        self._cf.log.add_config(self._lg_stab)
        # This callback will receive the data
        self._lg_stab.data_received_cb.add_callback(self._stab_log_data)
        # This callback will be called on errors
        self._lg_stab.error_cb.add_callback(self._stab_log_error)
        # Start the logging
        self._lg_stab.start()
    except KeyError as e:
        print('Could not start log configuration,'
              '{} not found in TOC'.format(str(e)))
    except AttributeError:
        print('Could not add Stabilizer log config, bad configuration.')
```

Per tant, per poder incorporar tan *logs*, *parameters* i *commanders*, el primer que vàrem provar va ser un programa senzill de `fall_safe`. Aquest programa no és més que una espècie de protecció, intentant així que en el moment en el que el dron caigués per qualsevol motiu, els motors reaccionessin oferint una empenta més elevada provocant que la caiguda fos de la forma més segura possible evitant grans xocs continuats amb el terra. Per tal d'aconseguir-ho, vàrem iniciar un *log* que ens proporcionava la mesura de l'acceleració vertical que obtenia en tot moment el sensor del dron cada 100 ms. Si aquesta

acceleració sobrepassava un llindar (determinat experimentalment) prèviament determinat, s'enviava un missatge *commander* per a que s'activés el protocol d'ús del programa de `fall safe` per tal d'evitar el xoc brusc amb el terra A la figura 2.14 es pot observar una de les proves en les que activàvem de forma manual el `fall safe` agafant el dron i simulant una caiguda.

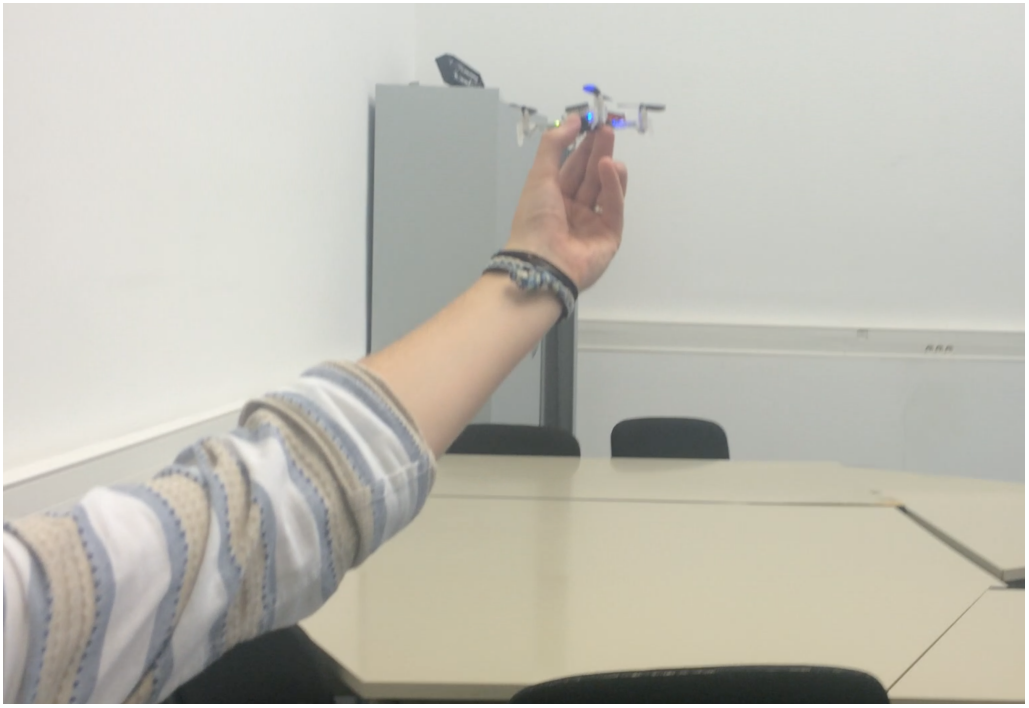

Figura 2.14: Proves del `fall_safe`

El principal problema que vàrem veure amb aquest programa és la necessitat de tenir funcions actuant de forma paral·lela, és a dir, amb diferents fils (*threads*). Per exemple en els primers intents, quan es passava el llindar dels *logs*, s'activaven els motors a una potencia del 60% i després es reduïa de forma gradual la potència fins a arribar a 0%. Ara bé, quan es rebia el primer *log* s'activava la potència, però com que el dron seguia caient, es rebia un altre *log* que tornava a activar els motors a 60%. Per tant, s'activaven molts `fall safe` i en cap moment es baixava la potència. Ens vàrem adonar doncs que necessitàvem un fil apart que actués de forma independent a la resta del sistema. Ho vàrem sol·lucionar amb la llibreria pròpia de Python anomenada `threading`. Amb aquests canvis, en rebre un *log* on l'acceleració superés el llindar, s'activava el `fall safe` i aquest activava els motors i fins que no havia acabat de fer les tasques pertinents no es tornava a activar en cas de seguir rebent els *logs*. Si bé és cert que els primers intents no van ser del tot satisfactoris i podríem definir-ho com que van fallar, ajustant paràmetres com per exemple el temps d'enviament dels *logs* a 100 ms i després de moltes proves i errors vàrem aconseguir que aquest petit sistema funcionés. A continuació el codi amb la implementació.

```
class ControllerCrazyflie:
    def __init__(self, link_uri):
        self.link_uri = link_uri
        self.cf = Crazyflie()
        self.cf.connected.add_callback(self.cf_connected)
        self.cf.disconnected.add_callback(self.cf_disconnected)
```

```
        self.cf.connection_failed.add_callback(self.cf_disconnected)
        self.cf.connection_lost.add_callback(self.cf_disconnected)
        self.acc_z = 0
        self.cf.open_link(link_uri)
        self.is_connected = True
        self.start_logs()
        self.fall_safe_activated = False

    def start_logs(self):
        # Create the log configuration with 100 ms interval
        self.log_control = LogConfig(name='Log_control', period_in_ms=100)
        self.log_control.add_variable('acc.z', 'float')
        self.log_control.data_received_cb.add_callback(self.log_data_received)
        self.log_control.error_cb.add_callback(self.log_error_received)

    def cf_connected(self):
        self.cf.log.add_config(self.log_control) # First add the logs to cf
        self.cf.commander.send_setpoint(0, 0, 0, 0) # Activate the motors
        self.log_control.start() # Start the logs receiver

    def cf_disconnected(self):
        self.cf.close_link()

    def log_data_received(self, timestamp, data, msg):
        # Update internal variable data with the received one
        self.acc_z = data['acc.z']
        # Fall safe activation with different thread
        if self.acc_z < 1.1 and self.fall_safe_activated is False:
            self.fall_safe_activated = True
            Thread(target=self.fall_safe).start()

    def log_error_received(self, logconf, msg):
        print('Error when logging %s: %s' % (logconf.name, msg))

    def set_thrust(self, percentage):
        # values of thrust between 0 - 65535
        temp_thrust = int((percentage / 100) * 65535)
        self.cf.commander.send_setpoint(0, 0, 0, temp_thrust)

    def fall_safe(self):
        ramp_thrust = 60
        while ramp_thrust >= 40:
            self.set_thrust(ramp_thrust)
            time.sleep(1)
            ramp_thrust -= 5
        self.set_thrust(0)
        self.fall_safe_activated = False
```

El programa de `graphic_fall_safe` és una variació del `fall_safe` en que es guarden les dades que s'estan rebent del *log* de l'acceleració i es fan gràfiques en temps real mitjançant la llibreria de Python `matplotlib`. L'avantatge d'aquest programa és que permet monitoritzar en tot moment quin és el valor de l'acceleració sense haver d'observar el dron. Tal i com es pot apreciar a les figura 2.15 següent en que es detecta la caiguda del dron.

```
import matplotlib.pyplot as plt
from drawnow import *
```

```
3
4  acc = []
5  plt.ion()
6  cnt = 0
7
8  def makeFig():  # Create a function that makes our desired plot
9      global acc
10     plt.ylim(-2, 2)                                    # Set y min and max values
11     plt.title('My Live Streaming Sensor Data')  # Plot the title
12     plt.grid(True)  # Turn the grid on
13     plt.ylabel('Acceleration on z axis')  # Set ylabels
14     plt.plot(acc, 'r-', label='Acceleration on Z')
15
16 def plot_data(acc_temp):
17     global cnt
18     global acc
19     acc.append(acc_temp)
20     drawnow(makeFig)
21     plt.pause(0.00001)
22     cnt += 1
23     if cnt > 10:  # If you have 50 or more points, delete the first one from
       the array
24         acc.pop(0)
```

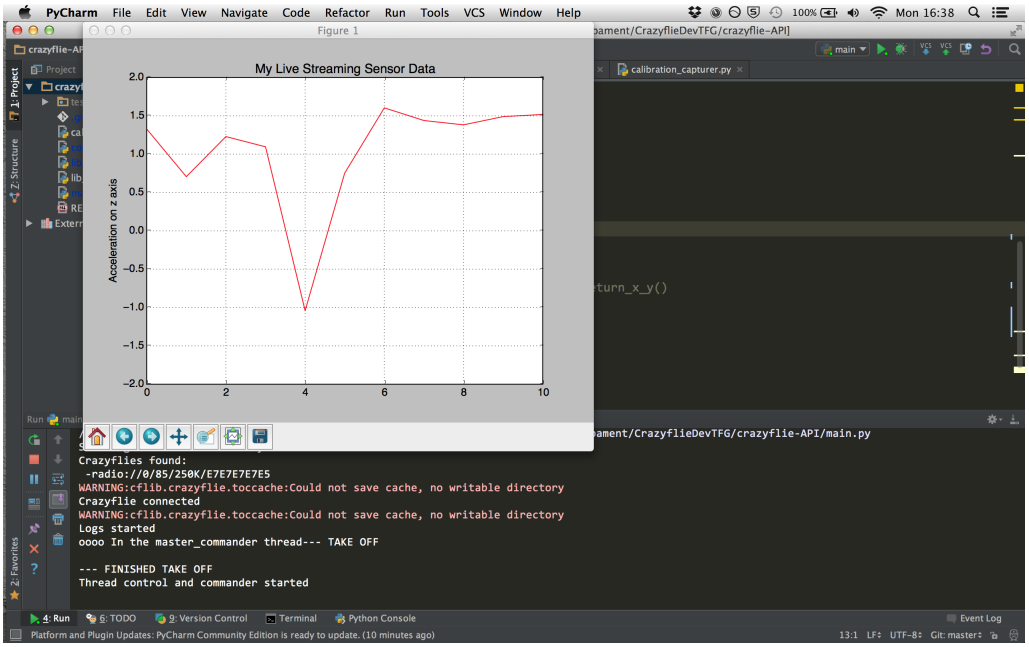


Figura 2.15: Gràfica del `graphic_fall_safe`

A continuació i per la necessitat de tenir algun tipus de de control autònom es va veure la necessitat d'implementar el `commander_thread`. Aquest programa és simplement una variació respecte als demés. El canvi en el funcionament és que ara en iniciar el Crazyflie, també s'iniciarà un *thread* separat que es dedicarà exclusivament a enviar *commander* amb informació de variables globals. Si en qualsevol dels altres *threads* es modifiquen els valors que s'envien amb el *commander*, aquest els enviarà immediatament. D'aquest manera, s'eviten les interferències en que cada mòdul envia paquets de *commander* que no tenen correlació. Per exemple, si volem programar un `take off` però el control de-

tecta uns canvis necessaris i també envia *commanders* provocant així que es tingui un vol irregular i erràtic.

```
def master_commander(self):
    # Activate the motors
    self.cf.commander.send_setpoint(0, 0, 0, 0)

    # Infinite loop sends paquets every 100 ms
    while True:
        self.cf.commander.send_setpoint(self.roll, self.stab_pitch + self.pitch
    , self.yaw, self.thrust)
        time.sleep(0.1)
```

Després de tenir ja el *commander* es va decidir crear un `control_thread`. Aquest primer crea un *log* separat del de l'acceleració del `fall_safe` que rep tres variables del Crazyflie, l'estabilització del *pitch*, del *roll* i del *yaw*. En el moment en que arriben, s'actualitzen els valors de les variables locals. S'inicia doncs un nou *thread* que segons el valor que rep modifica els valors que s'enviaran amb el `commander_thread` per tal que aquests valors siguin el més pròxim als valors de *trimatge* determinats de forma experimental amb els experiments usant el client de Python (taula 2.2).

```
def control_from_logs(self):
    value_rate = 2
    pitch_trim = -6
    roll_trim = 3

    if self.stab_pitch > pitch_trim:
        self.pitch -= value_rate
    elif self.stab_pitch < pitch_trim:
        self.pitch += value_rate

    if self.stab_roll > roll_trim:
        self.roll -= value_rate
    elif self.stab_roll < roll_trim:
        self.roll += value_rate
```

Finalment s'han integrats les millores de cada un dels programes en un programa principal anomenat `main_control`. De forma esquemàtica es pot apreciar el funcionament bàsic a la figura 2.16

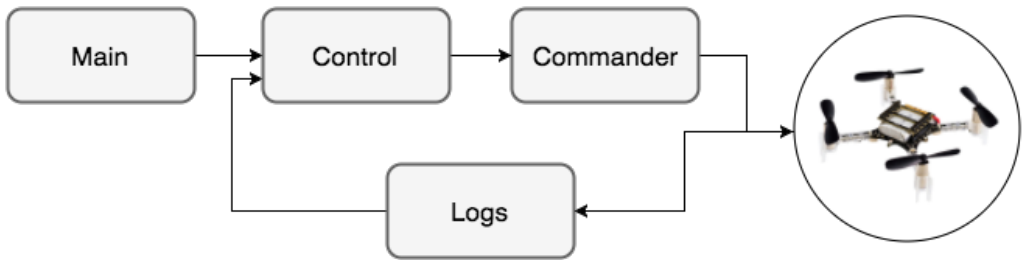


Figura 2.16: Esquemàtic de funcionament del `main_control`

Com bé es pot observar a la figura prèvia, s'han definit quatre mòduls principals dins del sistema de control:

- Main (principal): és la base de tot i el que inicialitzarà la resta de *threads*. A part d'això també donarà inici al vol amb un enlairament programat. A més a més,

també comptarà, entre d'altres, amb les funcions de *landing* (aterratge) i activarà el `fall_safe` en cas que el dron caigui sense control.

- Control: és l'apartat on es faran tots els càlculs necessaris per a poder controlar el vol del dron. Aquesta part de l'estudi té dues entrades, el programa principal i els *logs* provinents del Crazyflie com a dades ja enregistrades duran cada segon del vol. Usant totes aquestes dades s'obtindrà les dades del *pitch*, *roll*, *yaw* i *thrust* que posteriorment seran enviades al *commander*.
- Commander: aquesta secció s'encarrega de rebre la informació enviada pel control, genera els paquets segons les dades rebudes i a continuació els envia al Crazyflie.
- Logs: El Crazyflie envia de forma periòdica, mitjançant els sensors, l'estat del *pitch*, *roll*, *yaw* i acc_z al mòdul de Logs que actualitza la informació rebuda i la passa al control.

El codi del control es pot apreciar en el annexes amb detall de cada funció i mòdul implementat.

Tenint tots aquest mòduls, un dels aspectes més importants és el d'encaixar bé els temps. El que idealment es vol és rebre paquets, processar-los i enviar-los el més ràpid possible per tal de no produir cap *delay* en els següents mòduls. El problema però és que si s'envien massa paquets hi ha la possibilitat de que es perdi la connexió, no hi hagi temps de processar-los tots o fins i tot que es col·lapsi el Crazyflie.

Per tant s'ha de trobar una solució de compromís entre la latència que volem reduir i el mínim de processament que es pot realitzar. El problema de pèrdua de connexió és un dels primers problemes reals i importants al qual ens vam haver d'enfrontar. Al principi preteníem obtenir tota la possible informació que pot arribar a proporcionar el dron mitjançant *logs* però ràpidament vam veure la impossibilitat de realitzar-ho degut a la pèrdua constant de connexió amb el dron, fet que ens impossibilitava la realització de les proves de manera eficient. Per tant, per poder trobar una solució satisfactòria es va escollir els temps de transmissió de cada *log* fent un compromís entre velocitat, pèrdua de missatges i capacitat *computacional*.

A la taula 2.4, tenim en resum els temps d'enviament de comandes de cada un dels mòduls. Aquestes dades s'han descobert de forma experimental.

Taula 2.4: Temps de procés del `main_control`

| **Mòdul** | **Temps definit (ms)** |
|---|---|
| Log d'acceleració en z | 200 |
| Logs d'estabilització | 50 |
| Commander thread | 100 |

Com s'aprecia a la taula, s'ha donat prioritat i més rapidesa a l'estabilitat que no pas a la seguretat del `fall safe`, perquè en els experiments realitzats en aquest programa, l'alçada a la qual arribava el Crazyflie no era molt elevada. Així doncs, el risc de trencar-se el dron era menor en cas de connexió perduda.

En comparació amb les anteriors proves del client d'android i el client de Python s'ha pogut observar una optimització molt notable de la bateria (entre 8 i 9 minuts). Això és gràcies a

que s'està optimitzant l'ús dels motors per a mantenir una certa alçada. El fet de no variar el *thrust* i les direccions contínuament també és un factor determinant en l'autonomia del Crazyflie. Per tant amb un vol més eficient aconseguim augmentar l'autonomia del Crazyflie.

Amb aquest sistema de control s'ha pogut realitzar un enlairament continu, mantenir una certa alçada de forma estable un temps i finalment aterrar de forma controlada. Ara que ja es té un model de control del Crazyflie, l'únic que queda és afegir l'entrada del sistema de posicionament local per poder indicar cap a on ha d'anar i com s'ha de posicionar.

## 2.3. Parrot 2.0 ARDrone

Pel que fa als sistemes remots actius, l'element escollit per treballar ha estat el dron Parrot 2.0 AR Drone. El AR Drone és un quadrotor a control remot desenvolupat per Parrot SA. La principal diferència amb el Crazyflie, el dron estudiat amb anterioritat, és la mida (figura 2.17). En aquest segon cas la mida és molt més superior que en el primer cas.

Si ens fixem en els principals avantatges d'aquest quadrotor cal esmentar a simple vista la seva robustesa adquirida sobretot per la seva mida i proteccions i també el preu assequible (entre 200 i 300 €).

Figura 2.17: Parrot AR.Drone 2.0

El vehicle es controla mitjançant l'enviament de comandes a través d'una connexió Wi-Fi i l'ús d'una aplicació necessària. Per altre banda el AR Drone està equipat amb una càmera frontal i una altre situada a la part inferior del dron que enfoca cap avall. Ambdues càmeres ofereixen la transmissió de vídeo en directe. A més, dos algoritmes complementaris fan ús de la càmera inferior fet que proporciona una millor estabilització. D'altra banda, el AR.Drone està equipat amb un sensor d'ultrasons i una unitat inercial que mesura entre d'altres característiques el capcineig (*pitch*), el balanceig (*roll*), la guinyada (*yaw*) i l'acceleració al llarg de tots els eixos (figura 2.18).

### 2.3.1. Control del Parrot

El primer que cal fer és definir l'estructura mecànica del Parrot. Aquest dron es compon de quatre rotors units al cos central del quadrtor. Cada parell de rotors oposats, és a dir el 1 i 3 i el 2 i el 4, giren en la mateixa direcció. Una altre de les característiques és que un parell gira en sentit horari i l'altre en sentit anti-horari. Cada rotor produeix tant un *thrust* T com un parell (*torque*) $\tau$ al voltant del seu centre de rotació, així com un *drag* D oposat a la direcció de vol del vehicle.

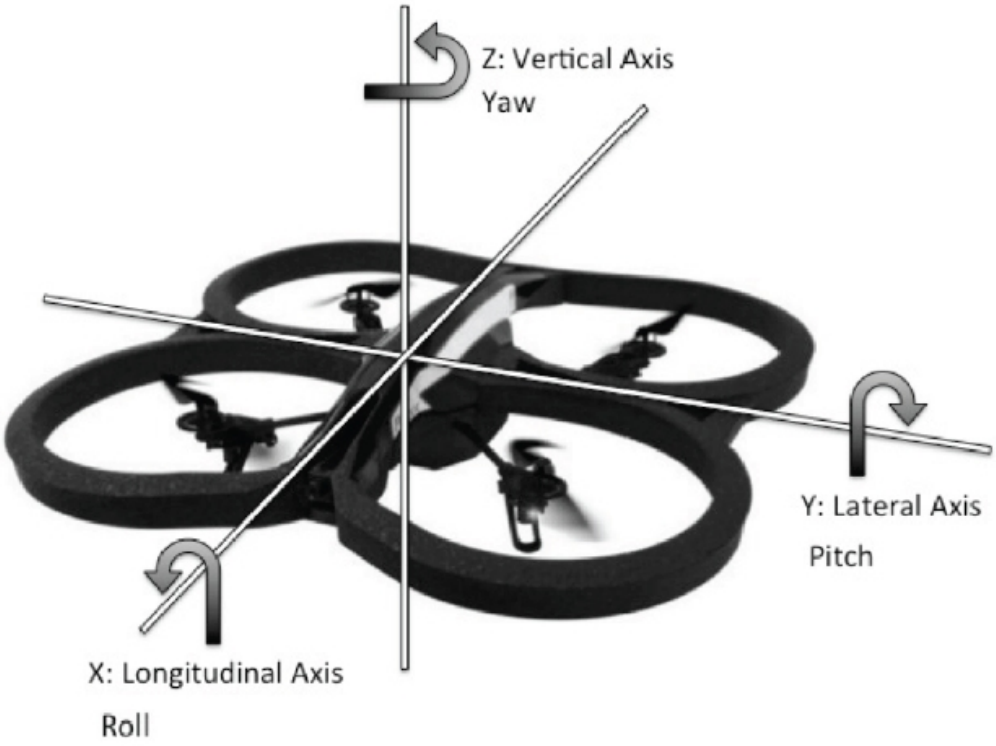


Figura 2.18: Diferents moviments al voltant del eixos principals

El fet que tots els rotors estiguin girant a la mateixa velocitat angular, provoca que el parell diferencial sigui zero, que resulta en acceleració angular nul·la al voltant de l'eix de guinyada. Per tant, la variació de la velocitat angular de cada parell rotor produeix una acceleració angular al voltant de l'eix de guinyada. Per altra banda també sabem que si l'empenta total (*thrust*) és manté constant, la velocitat vertical també s'hi mantindrà. Un moviment vertical s'aconsegueix augmentant o disminuint l'empenta de cada motor a la mateixa quantitat, de manera que el dron acaba sofrint canvis en l'empenta total però el parell diferencial en el cos segueix sent zero. Per tal d'obtenir un moviment horitzontal el que cal fer és senzillament mantenir un angle de balanceig (*roll*) o de capcineig (*pitch*) diferent a zero.

### 2.3.2. Hardware del Parrot

Com s'ha comentat amb anterioritat, el Parrot es tracta d'un quadrotor. Així doncs, consta de quatre motors sense escombretes, com en el cas del Crazyflie, que estan units al cos principal del dron. Aquest cos principal està constituït d'una estructura de tub de fibra de

carboni i plàstic d'alta resistència. El cos principal, al ser la base de tot el dron, consta d'una protecció que anomenarem casc, degut a que encaixa perfectament amb la part superior del cos. El casc de protecció està fet del material d'escuma 3 de polipropilè expandit (EPP), que com a característiques principals es podria dir que és resistent, de pes lleuger i reciclable. El casc ofereix protecció durant els vols en interiors, fet que en el nostre cas és molt important per no patir danys seriosos. Les hèlixs estan alimentades per energia proporcionada per una bateria de polímer de liti que consta d'una capacitat de 1000 mAh, el que permet un temps de vol d'aproximadament 10 minuts a ple rendiment.

En quant al processador incorporat al AR Drone cal parlar del model ARM9 amb 468 MHz. Un processador de la mateixa casa que en cas del Crazyflie. Aquest model té una RAM de 128 MB que treballa amb un sistema operatiu Linux personalitzat. A l'hora de realitzar el control no hi ha cap comandament creat per aquest propòsit, sinó que qualsevol dispositiu ordinari amb Wi-Fi es pot utilitzar per controlar el AR.Drone (usualment un mòbil o Tablet). Cal destacar que inicialment l'aplicació encarregada de controlar el Parrot, anomenada AR.FreeFlight 2.4.15 i que es pot trobar al Google Store de qualsevol dispositiu mòbil, estava pensada exclusivament per plataforma Apple però des de 2011 ja va ser possible adquirir-la per les altres plataformes. Per controlar el dron des d'una plataforma Windows o Linux (ordinador) és necessari obtenir el *software* desenvolupat pels dissenyadors de l'aplicació.

#### *2.3.2.1. Subsistemes inercials de Parrot*

La unitat de mesura es tracta d'un sistema micro electro-mecànic (MEMS) el qual conté un acceleròmetre de 3 eixos, un giròscop de 2 eixos (balanceig i capcineig) i un giroscopi d'un sol eix, el de guinyada.

L'acceleròmetre es tracta d'un model de la casa Bosch Sensortec en concret el BMA150, i s'encarrega de mostrar la força g de la gravetat com una acceleració basant-se en la teoria que el pes de la massa varia durant l'acceleració. El Parrot AR Drone també està composat per dos giròscops. Cal apuntar que un giroscopi és un aparell capaç de mesurar velocitats angulars basant-se en els principis dels moments angulars. Per tal d'obtenir l'angle amb el qual treballen s'integrarà en funció del temps les velocitats angulars obtingudes durant les mesures. El problema d'aquest mètode són els errors de biaix o de parcialitat (*bias errors*) fet gairebé impossible d'eradicar en qualsevol aparell d'aquest tipus.

El model usat en aquest dron pel giroscopi de dos eixos és el giroscopi IDG-500 Dual-Axis5 caracteritzat per tenir dues sortides separades per cada eix, una per als moviments de major velocitats i l'altre per moviments més precisos i lents. D'altra banda, per al giròscop de l'eix de guinyada s'usa el Epson Toyocom XV-3500CB un giroscopi d'alta precisió. Les característiques principals estan a la taula 2.5

Taula 2.5: Específicacions dels giròscops del Parrot

| **Model** | **Rang** | **Sensibilitat** |
|---|---|---|
| IDG-500 moviments altes velocitats | 500 $^{o}$/s | 2 mV/$^{o}$/s |
| IDG-500 moviments baixes velocitats | 110 $^{o}$/s | 9,1 mV/$^{o}$/s |
| Epson Toyocom XV-3500CB | 100 $^{o}$/s | 0,67 mV/$^{o}$/s |

#### *2.3.2.2. Càmeres del Parrot*

Com bé s'ha esmentat en el primer apartat d'aquest capítol, l'AR.Drone està equipat amb dues càmeres CMOS (de l'anglès *complementary metal oxide semiconductor*), una càmera frontal i una càmera situada a la part inferior del cos central. Les dues càmeres permeten i, de fet, executen la transmissió de vídeo en directe a 15 fotogrames per segon. La diferència entre ambdues càmeres (taula 2.6) és que la càmera frontal s'utilitza normalment per detectar altres drons, obstacles o bé per proporcionar informació visual per pantalla en forma de vídeo mitjançant l'aplicació de control del dron. La freqüència de vídeo és de 60 *frames* per segon per tal de reduir el desenfocament per moviment. En canvi, la càmera inferior s'utilitza principalment per a l'estabilització horitzontal i per estimar la velocitat del dron. Així doncs, es pot afirmar que aquesta càmera inferior juga un paper molt important quan es parla de la intel·ligència a bord del Parrot AR.Drone.

Taula 2.6: Específicacions de les càmeres del Parrot

| | **Resolució** | **Angle de captura** |
|---|---|---|
| Càmera Frontal | 640 x 480 píxels (VGA) | $93^o$ |
| Càmera Inferior | 176 x 144 píxels (QCIF) | $64^o$ |

### 2.3.3. Software del Parrot

El *software* desenvolupat per aquest dron està escrit en codi C i es troba dintre el AR.Drone API (de l'anglès *Application Programming Interface*) que es tracta del projecte referència en termes de desenvolupament d'aplicacions d'aquesta empresa.

El funcionament és força semblant a l'explicat al Crazyflie. El control i configuració del dron es fa mitjançant *commands* de manera regular que en aquest cas s'anomenen `AT commands`. En canvi, la informació rebuda o proporcionada al client pel dron, ja sigui altitud, velocitat, estat de l'aparell, etc... s'anomena `navdata`. Aquesta també inclou diferents dades de mesura proporcionades pels sensors. Per últim, però no menys important, tenim l'apartat del vídeo. El vídeo en *stream* s'envia per part del dron al client, i per tal de descodificar les imatges d'aquest vídeo, s'usen uns descodificadors inclosos en el *Software de Desenvolupament* (SDK).

D'altra banda també es troba la AR.Drone Library que com bé indica el seu nom, és una llibreria inclosa en el *software* que es pot dividir en tres seccions diferenciades:

- SOFT, llibreria encarregada principalment de l'estructura de comunicació.
- VPSDK, llibreries encarregades de tota mena de propòsits diferents.
- VLIB, llibreria que s'encarrega de processar el vídeo.

Desafortunadament, com bé s'ha comentat amb anterioritat, l'aplicació estava principalment pensada per Apple i això ha causat que l'ús per altres plataformes requereixi de la solució de certs problemes que s'ocasionen al usar-la.

En el nostre cas en concret es va escollir usar la llibreria `python-venthur` extreta de la pagina web amb el seu corresponent Git [17]. Ens vam trobar amb diferents llibreries

que executaven el mateix de formes similars, però poques estaven escrites per llenguatge Python i l'esmentada prèviament va semblar la més robusta i fiable a l'hora de provar-la. Com s'ha comentat però, existeixen alguns inconvenients en l'ús de l'aplicació per dispositius diferents a Apple, i en el nostre cas ens els vam trobar, sent completament incapaços de solucionar-los o de trobar una solució alternativa. Hem de dir que en el control i manipulació del dron no vam tenir més problemes que el d'aprendre a fer-lo funcionar i volar, però a l'hora de treballar amb les càmeres i el vídeo proporcionat per aquestes, les coses van ser molt diferents. Si bé es cert que la càmera frontal funcionava correctament ja que usant l'aplicació mòbil obteníem les imatges ofertes per aquesta, usant les llibreries per ordinador no vam ser capaços de cap de les maneres d'obtenir les imatges (el vídeo) que ens havia de proporcionar aquesta càmera, resultant així inútil l'ús d'aquest dron pels nostres propòsits inicials.

Arribats a aquest punt del treball cal aclarir un punt important. Tot l'estudi i experiments realitzats al Crazyflie es podria haver dut a terme exactament igual sobre el Parrot, una vegada ens vam topar amb la impossibilitat de rebre les imatges per part d'aquest dron. El principal motiu pel qual no es va executar doncs, és perquè la nostre idea inicial era la de treballar amb un dron en el qual s'estudiés els elements passius i un altre amb els actius. Es va escollir al Parrot com a element actiu perquè la càmera que porta incorporada el fa especialment atractiu per aquest propòsit. Així doncs, i veient que la estabilització del Parrot era molt fàcil de realitzar i manipular i que no suposava cap repte de cara a un estudi d'aquestes característiques, es va decidir treballar el control del Crazyflie. De totes maneres, com s'ha comentat prèviament, en altres estudis on els propòsits no siguin exactament els mateixos que en el nostre cas, es podria adaptar tot el Parrot al treball fet amb el Crazyflie amb molta facilitat només amb la necessitat de variar certs paràmetres del *trimatge*, de magnituds d'empenta inicial i també la llibreria de control que passaria a ser l'anomendada `python-venthur` en comptes de la llibreria del Crazyflie.

# CAPÍTOL 3. VISIÓ PER COMPUTADORS

En aquest primer punt del capítol es tractarà el tema de l'obtenció de la posició del dron mitjançant el nostre sistema de posicionament local escollit: les càmeres. El camp de la visió per computador és extens, amb molts matisos i molt desenvolupat actualment. Dins de les moltes llibreries que es van arribar a consultar, es va escollir finalment opencv (figura 3.1).


Figura 3.1: Logotip de la llibreria OpenCV

Aquesta llibreria [18] a més a més de ser molt completa, té un avantatge principal i és que es pot integrar des del llenguatge de programació ja usat per el Crazyflie, és a dir, que podem cridar a la llibreria des de Python. El fet d'haver fet tot el desenvolupament del Crazyflie mitjançant Python, usar el mateix llenguatge i evitar problemes de concordança ens ha semblat l'opció més correcta i adequada. Així doncs, una vegada escollida la llibreria es va instal·lar amb les dependències associades. Una vegada feta la instal·lació es pot començar amb el desenvolupament d'aquest mòdul de posicionament local.

El primer que cal és detectar una càmera. En el nostre cas s'usarà una càmera USB ja que principalment és l'opció que ens va proporcionar el tutor, és intuïtiva, fàcil d'implementar i funciona des de la primera prova. Tenint en compte que hi havia molts altres problemes, s'ha considerat que provar altres opcions tenint la seguretat que d'aquesta manera funciona tot correctament és complicar les coses innecessàriament. Un cop registrada la càmera, el que farem serà anar capturant cada *frame*/imatge de la càmera i es processarà individualment. No processarem el vídeo que es registri com a un conjunt tancat sinó que s'analitzarà imatge a imatge, ja que la nostra idea és poder analitzar i determinar en temps real la posició del dron amb el que realitzem el nostre estudi. És per aquest motiu que el processament que es faci de cada *frame* no pot ser molt elaborat i complicat ja que limitaria la resposta que tinguem del control.

Per poder detectar el nostre dron, en aquest cas el Crazyflie, hem d'aconseguir que la nostre càmera sigui capaç de trobar alguna singularitat única en l'objecte a detectar. En el nostre estudi s'ha treballat amb diferents possibles característiques que vam pensar que serien relativament fàcils de detectar mitjançant la càmera i l'ordinador.

Un cop feta la detecció, el següent pas és buscar la forma d'implementar algoritmes que ens permetin determinar la posició exacta en un espai tancat.

## 3.1. Detecció de colors i emmascarat

Per a poder detectar alguna característica que fos única del Crazyflie, inspirats en varis estudis, primerament vàrem decidir fer una detecció de colors. La idea principal és bastant senzilla: definir un interval de valors determinat a una imatge i, a continuació, aplicar-hi una màscara que agafa els colors que estiguin dins d'aquest interval definit. Cal comentar que cada tonalitat de color té un valor assignat i que per tant, contra més gran sigui aquest interval més tons del color s'agafarà. De la mateixa manera, si l'interval indicat de colors és molt estret, és possible que no es detecti cap color o bé que només se'n detecti un amb una saturació molt especifica i inamovible. Els colors es poden definir de moltes maneres. En aquest cas s'ha escollit definir-los com a HSV (de l'anglès *Hue, Saturation and Value*).

Segons la definició:

- Hue és la gamma del color
- Saturation és quant d'aquest color es té, la densitat d'aquest color per dir-ho d'una altra manera.
- Value es pot entendre com la lluminositat.

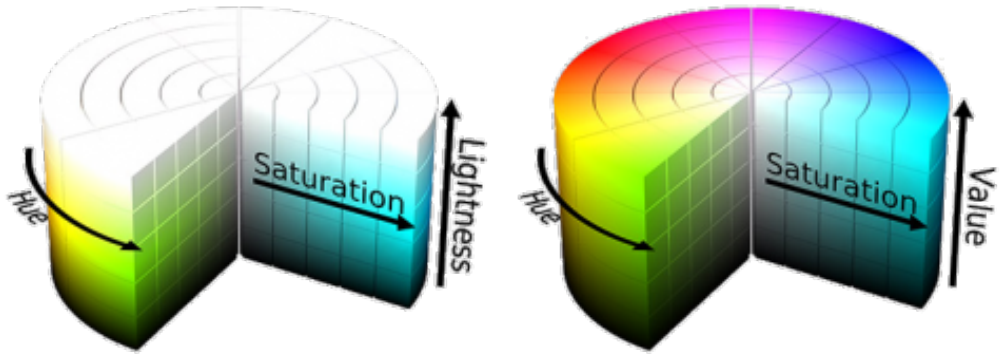


Figura 3.2: Nomenclatura dels colors segons HSL o HSV

Per tal de poder determinar un color característic, fent varies proves de detecció i veient diferents exemples existents, es va arribar a la conclusió que el color vermell era un bona opció ja que és un color que ressalta molt amb fons blancs i que no varia gaire amb la lluminositat. Per tant, vam posar un element vermell adjuntat al dron per tal de poder distingir-lo. Aquest element es tracta d'una pilota petita d'espuma que va ser situada a la part superior del Crazyflie (figura 3.3, figura 3.4.

Les dimensions d'aquesta pilota són òptimes ja que té un tamany suficientment gran com per eliminar fàcilment els possibles errors de detecció i també ens permet detectar el dron sense problemes a la distància de la càmera a la que es realitza el vol. Més que el tamany, el que ens importarà en aquest cas serà el color característic de la pilota, fet que serà el principal detall pel qual detectarem la pilota i no pas pel seu tamany. El pes del material podria ocasionar algun problema a l'estabilitat, però al ser d'espuma, ens permet seguir controlant el dron només reajustant els valors de *trimatge* i de *thrust* inicial. Per últim comentar que la pilota amb les característiques anteriors també serveix com a element de

mesura de seguretat ja que el fet que sigui d'espuma fa que amorteixi els cops bruscos que es puguin ocasionar impedint així un trencament de les peces del dron.

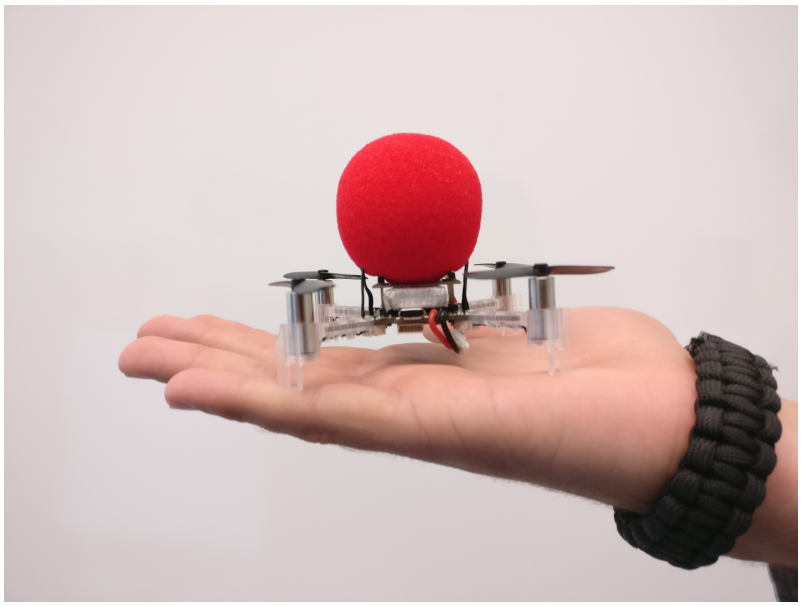

Figura 3.3: Crazyflie amb element de detecció

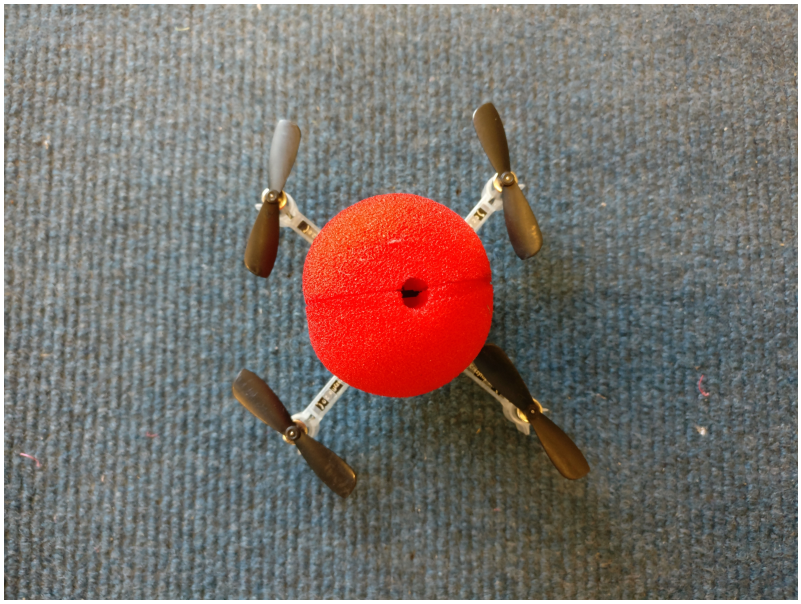

Figura 3.4: Crazyflie amb element de detecció amb vista vertical

Es van fer varies cal·libracions del color per tal d'ajustar correctament aquest interval de vermell tenint en compte la lluminositat com es pot apreciar en els annexes. En el nostre cas hem decidit agafar l'interval següent:

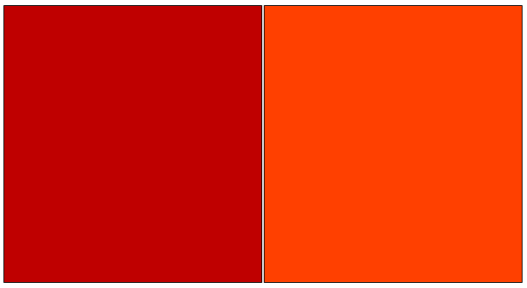

Figura 3.5: Paleta dels colors usats

**lower red =[0, 100, 75] upper red = [15, 255, 255]**

Després d'aplicar la màscara s'obté una imatge amb blanc i negre on apareix el color filtrat en blanc, en el nostre cas com s'ha comentat el vermell, i tota la resta d'entorn en negre. A la següent figura es pot apreciar amb claredat (figura 3.6).

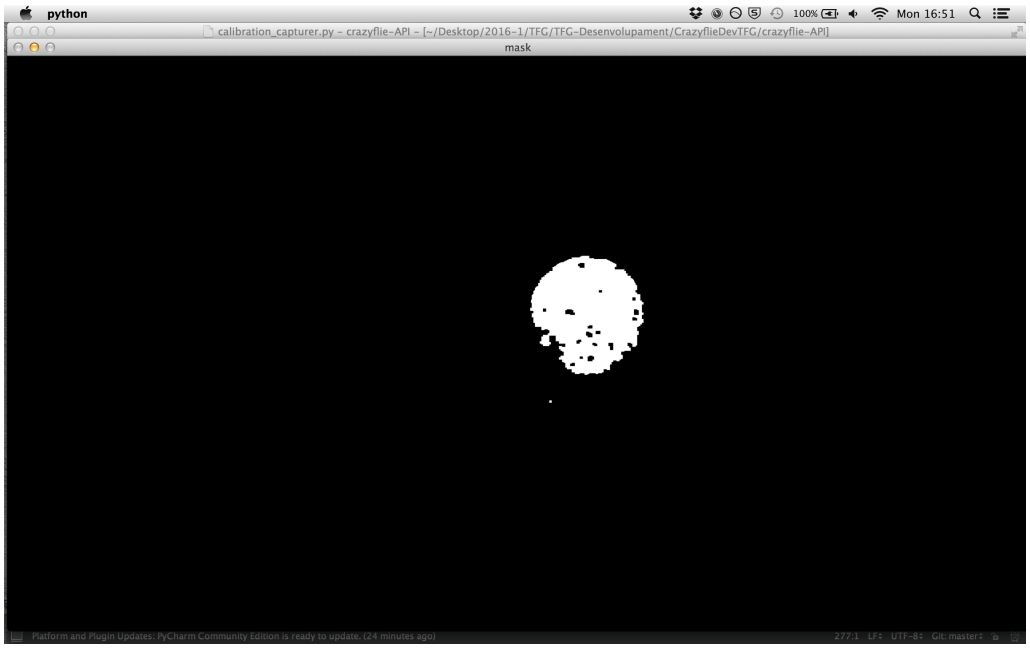


Figura 3.6: Mascara aplicada al nostre objecte

Per últim es fa un procés conegut com a *opening* i *closing* dels píxels, que bàsicament el que s'aconsegueix és arrodonir i reduir les interferències produïdes sobretot per altres colors propers a l'escollit i per altres factors. En les següents imatges es pot observar tots els passos anteriorment esmentats (figura 3.7).

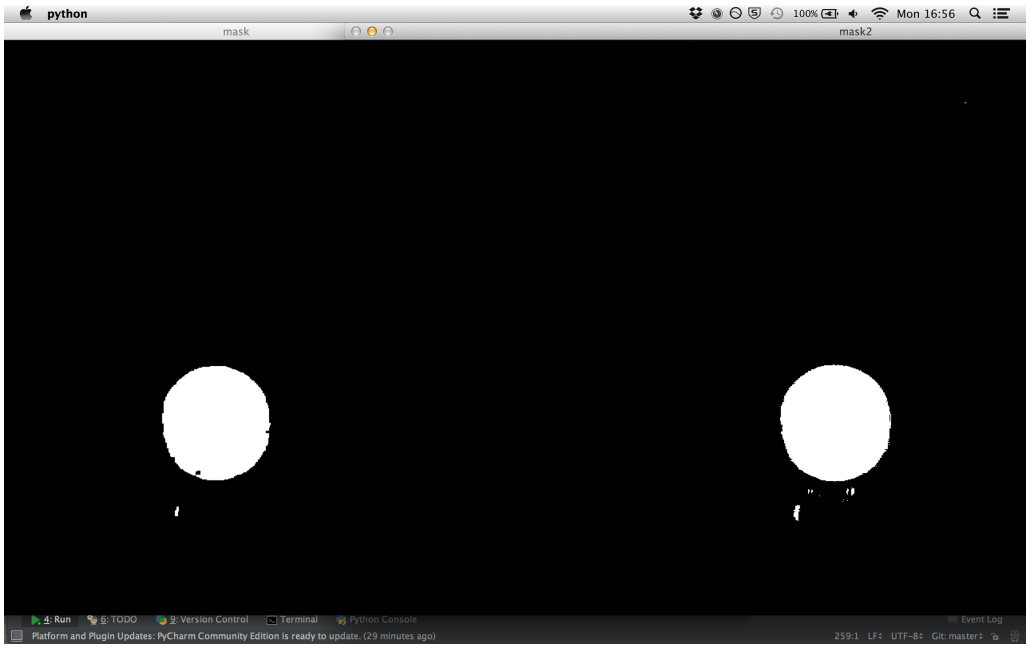


Figura 3.7: Reducció de les interferències de la nostra màscara

Els píxels que estan més junts es fan més grans i els píxels que estan més separats o sols es fan més petits per tal de minimitzar les interferències. D'aquesta forma s'aconsegueix tenir un objecte clarament més gran i definit que els demés, l'objecte a rastrejar. Seleccionant l'objecte amb una superfície més gran tindrem determinat ja l'objecte (figura 3.8).

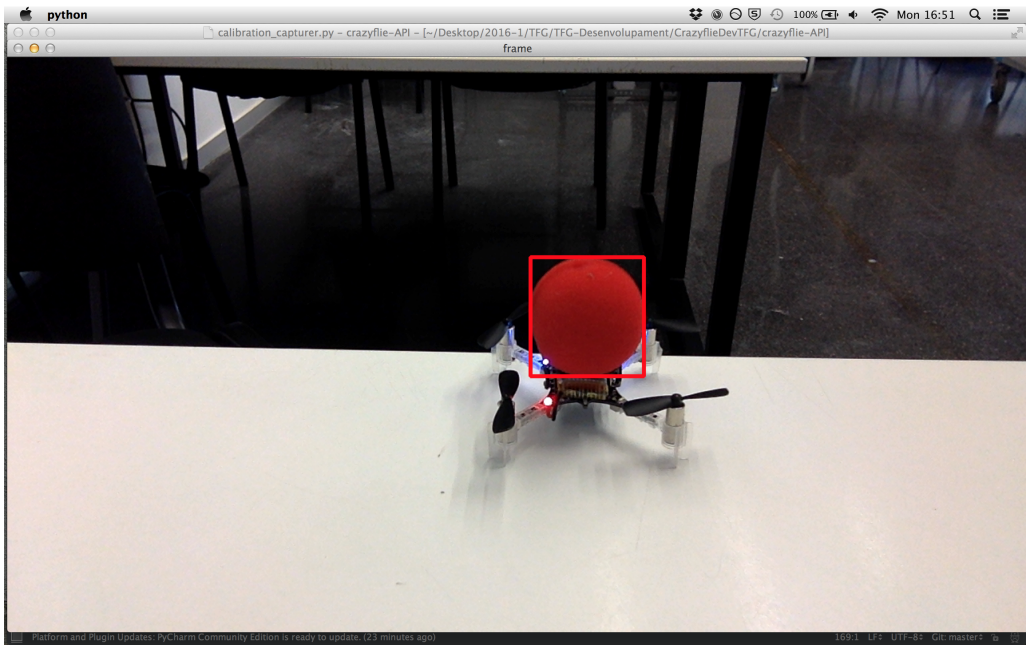


Figura 3.8: Exemple de detecció del color amb el dron

Per últim comentar que en el codi es veuen reflectits tots els passos a la funció que s'anomena `color_filter`.

## 3.2. Detecció de formes

Una altre de les singularitats de l'objecte a detectar pot ser directament alguna forma característica del dron. Si bé és cert que el filtre de color ens permet determinar la posició d'un color respecte una càmera fixa en dues dimensions, es pot determinar si l'objecte està situat amunt o avall i a la dreta o esquerra del punt d'enfocament de la càmera, també contempla alguns problemes que ens fan pensar en altres mètodes per detectar el nostre objectiu. El nostre problema principal usant el filtre de color apareix en dos situacions. La primera es dóna en el cas de que canviïn les condicions de lluminositat mentre que la segona situació apareix quan en el fons on estem treballant hi ha colors semblants al color de l'objectiu. D'aquest manera es pot afirmar que aquest sistema no és totalment fiable. El gran avantatge que té aquest mètode però, és que *computacionalment* és força ràpid (entre 3 i 6 *frames* per segon). Buscant altres alternatives es va trobar informació sobre els filtres de formes i d'aquesta manera es va provar d'implementar-ne un. Aprofitant que per poder detectar el color es va usar una esfera vermella (pilota petita) i es va decidir provar de detectar formes senzilles com cercles.

L'algoritme usat per detectar formes s'anomena `houghCircles` [19] i està inclòs dins de la llibreria d'opencv. Modificant certs paràmetres d'aquesta funció es poden detectar cercles d'una certa mida dins d'una imatge.

Aquests paràmetres inclouen:

- **image**:imatge de 8 bits convertit prèviament a escala de grisos
- **method**:Defineix el mètode de detecció dels cercles. Actualment només hi ha implementat el mètode `cv2.HOUGH_GRADIENT` [20]
- **minDist**:Mínima distància entre els centres dels cercles detectats. Si la distància és molt petita, és possible que es detectin massa cercles, en canvi si és molt gran podria no detectar-se'n cap.
- **param1**:Paràmetre que s'utilitza per determinar contorns.
- **param2**:És un limit llindar del numero de cercles que es detectaran. Si té un valor petit, més cercles es detectaran incloent falsos cercles. En canvi si és molt gran, és possible que es descartin cercles vertaders.
- **minRadius**:Mínim tamany del radi del cercle en píxels.
- **maxRadius**:Màxim tamany del radi del cercle en píxels.

Els valors escollit es poden apreciar a la taula següent 3.1:

Taula 3.1: Paràmetres de la funció houghCircles

| Paràmetre | Valor |
|---|---|
| minDist | 200 |
| param1 | 22 |
| param2 | 50 |
| minRadius | 50 |
| maxRadius | 100 |

Com en el cas anterior, també cal marcar uns límits que en aquest cas seran els del tamany del radi, així com marcar altres paràmetres ja definits en l'algoritme que ajudaran i delimitaran la detecció de les formes. El resultat es pot observar a la figura següent (figura 3.9).

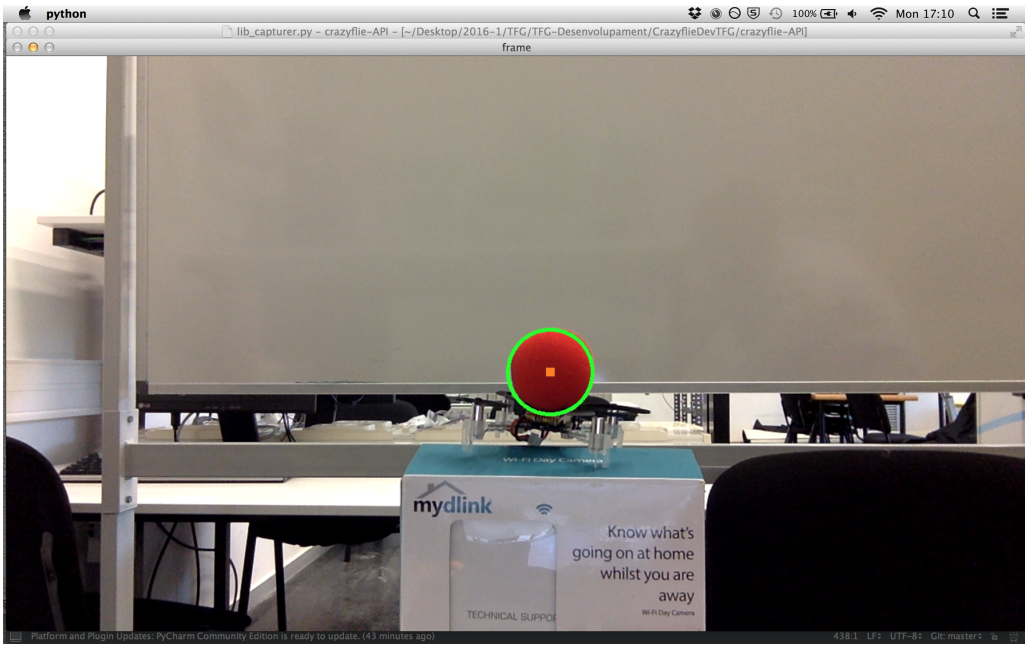


Figura 3.9: Exemple de detecció del cercle amb el dron

Com no podia ser d'una altra manera, aquest mètode també comporta un cert número de problemes o inconvenients. Un dels problemes principals d'aquest mètode és el temps de resposta. El temps necessari per detectar els cercles (1 ó 2 *frames* per segon) és massa elevat tenint en compte que nosaltres necessitarem informació en un període molt curt de temps i a més a més amb molta exactitud. El segon problema ve per aquí. La informació proporcionada pel programa no és sempre certa i encara menys precisa. Es dóna la situació en que es detecten cercles on simplement hi ha cúmuls de píxels semblants, mentre que a vegades entre imatge i imatge es perd el *tracking* de l'objecte.

Aquests errors es poden observar a la figura (figura 3.10).

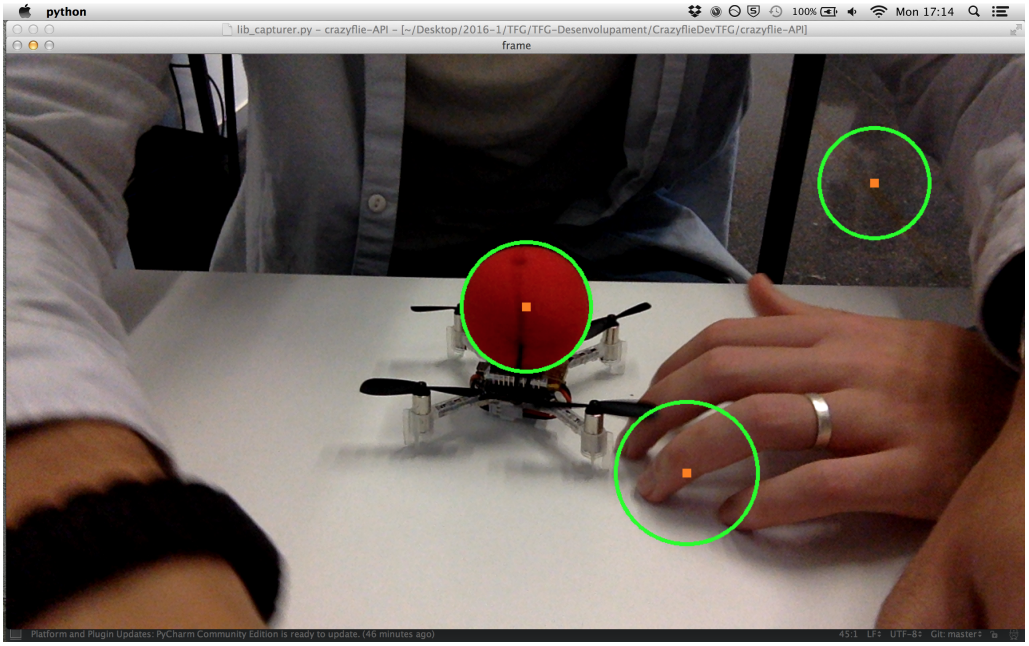


Figura 3.10: Errors obtinguts duran la detecció de cercles

En un principi es va intentar combinar les dues tècniques ja que així les possibilitats de detecció únicament del nostre dron augmentaven (contra més filtres posem més acurada serà la detecció de l'objecte al final), però ens vam trobar amb què les dues tècniques no es complementaven gens bé i en comptes de suplir els errors d'una i altre respectivament s'acumulaven i s'empitjorava la integritat del sistema. Es processaven només 1 *frame* per segon.

## 3.3. Detecció de moviment

Una vegada havent estudiat els dos mètodes prèviament explicats, es va decidir estudiar un últim mètode. Es va escollir indagar sobre un mètode inclòs dintre de la visió per computador, la detecció de moviment.

Tot i el fet d'usar una càmera estàtica i de poder afirmar, de forma general, que el fons dels experiments roman constant, una opció que se'ns va acudir que podria arribar a ser factible és la de detectar l'únic cos que està en moviment, és a dir el nostre objecte d'estudi, el dron. Informant-nos vam observar com hi ha molts algoritmes de detecció de moviment, però el que es va decidir utilitzar va ser un basat en l'eliminació de fons. Concretament l'algoritme usat s'anomena `BackgroundSubtractorMOG2` i es basa en una barreja-gaussiana [21].

Bàsicament el que fa aquest algoritme és comparar els píxels del *frame* actual amb l'anterior i allà on hi ha diferències genera una màscara semblant a la màscara del filtre de color. Un cop obtinguda aquesta mascara inicial, es processa de la mateixa manera que ho vàrem fer amb el mètode del filtrat de color. Se li apliquen els algoritmes de *closing* i *opening* per dilatar i erosionar les imperfeccions de la màscara i també per obtenir formes més homogènies. Finalment es selecciona la figura que tingui més superfície i aquesta serà el resultat del *tracking* mitjançant detecció de moviment. En les següents imatges es

poden apreciar un exemple ben clar d'aquesta funció de reducció de fons.

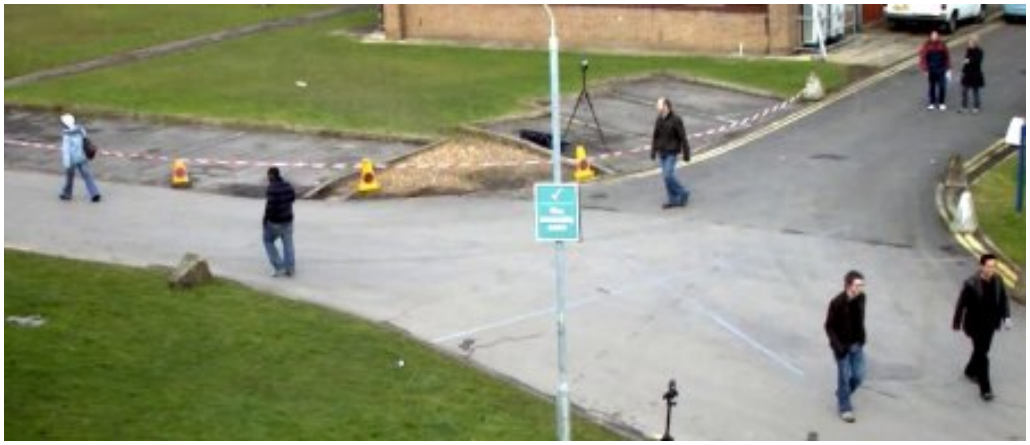

Figura 3.11: Exemple de detecció de moviment

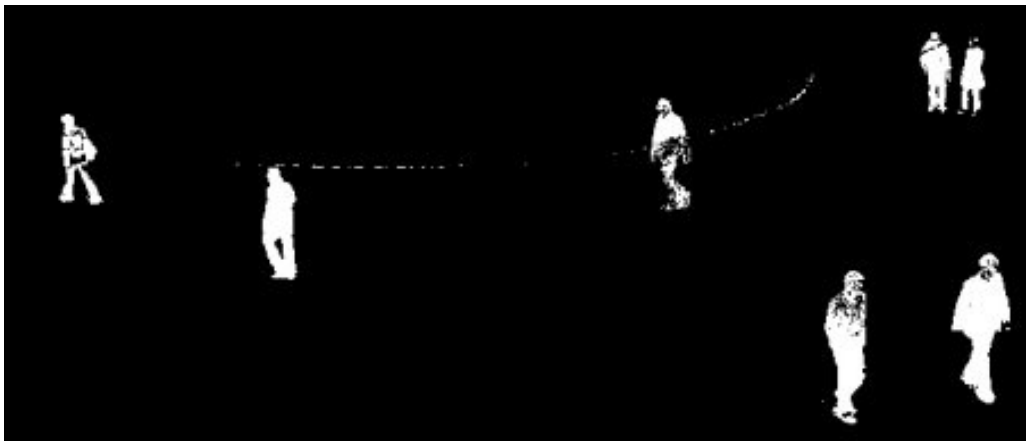

Figura 3.12: Màscara de l'exemple anterior

Una vegada la teoria estava clara, es van començar a fer proves. Cal destacar que en la majoria de les proves fetes amb el Crazyflie obteníem bons resultat. De totes maneres, com en tots els casos que hem estudiat té certs inconvenients. La limitació en aquest cas que presenta aquest mètode és que la càmera ha d'estar ben estàtica ja que qualsevol lleu moviment pot provocar grans canvis en la mesura de detecció del Crazyflie. Només amb la translació d'un píxel de la càmera aquest sistema dóna error momentàniament. Es va observar també com qualsevol moviment que provoqués una ombra en el fons, era detectat i suposava grans variacions i desajustos en les nostres mesures. En les següents imatges podem observar la detecció del Crazyflie mitjançant un rectangle verd usat en les nostres proves (figura 3.13).

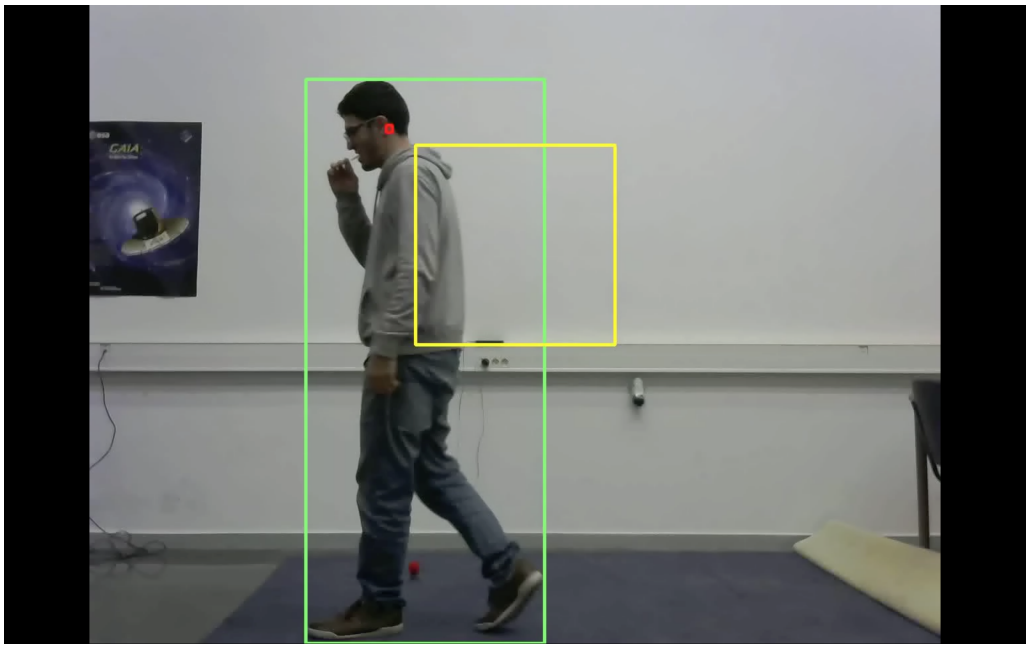


Figura 3.13: Exemple de detecció de moviment en el nostre estudi

## 3.4. Alternatives

A més a més dels diferents sistemes comentats prèviament, existeixen moltes alternatives als mètodes esmentats. El problema principal de totes aquestes alternatives no descrites és que són més complexes i per tant, el temps d'implementació és molt superior al temps limitat d'aquest projecte.

Entre totes aquestes alternatives, la que es podria destacar com més interessant seria sobretot la d'implementar *machine learning* en la detecció de l'objecte. La idea base d'aquesta opció és que amb un entrenament previ, és a dir, introduint com a *input* del mecanisme moltes fotografies del dron des de diferents angles i perspectives, l'ordinador sigui capaç d'entendre i detectar el dron degut a la informació (*inputs*) prèviament introduïda i analitzada. Per dir-ho d'una altra manera detectar el dron perquè ho ha "aprés". El *machine learning* és una de les ciències i metodologies amb un gran auge d'estudi per totes les possibilitats que pot oferir. Una de les llibreries de Python que vàrem estar investigant és `scikit-learn` [22] que juntament amb `scikit-image`, [23] una llibreria molt semblant a opencv, podrien haver estat la solució òptima a implementar pel nostre estudi. Finalment però, aquestes opcions van ser descartades. Aquestes llibreries basades en la idea de *machine learning* són gratuïtes i s'han implementat, com s'ha comentat, per Python entre d'altres. Com a característica també cal destacar el fet que es tracta d'un *software* de codi obert, simple i força eficient per l'anàlisi de dades. A més a més del temps d'implementació, una limitació important que suposava aquest tipus de sistemes és el temps de processat. Per a què fos un posicionament en temps real es necessitarien ordinadors amb processadors potents, aparells que no estaven dintre del nostre abast.

## 3.5. Posicionament mitjançant CV

Una cop estudiats i implementats diversos sistemes de detecció, el pas que queda és determinar la posició mitjançant l'anàlisi de la imatge que es captura usant aquests sistemes.

Inicialment el primer que es va decidir fer va ser determinar la posició horitzontal (x) i vertical (y) del objecte respecte la imatge que s'estava obtenint. Aquest objecte es tracta del nostre dron que ja el podíem detectar mitjançant els sistemes implementats. Es pot apreciar la situació prèviament descrita a la imatge següent (figura 3.14):

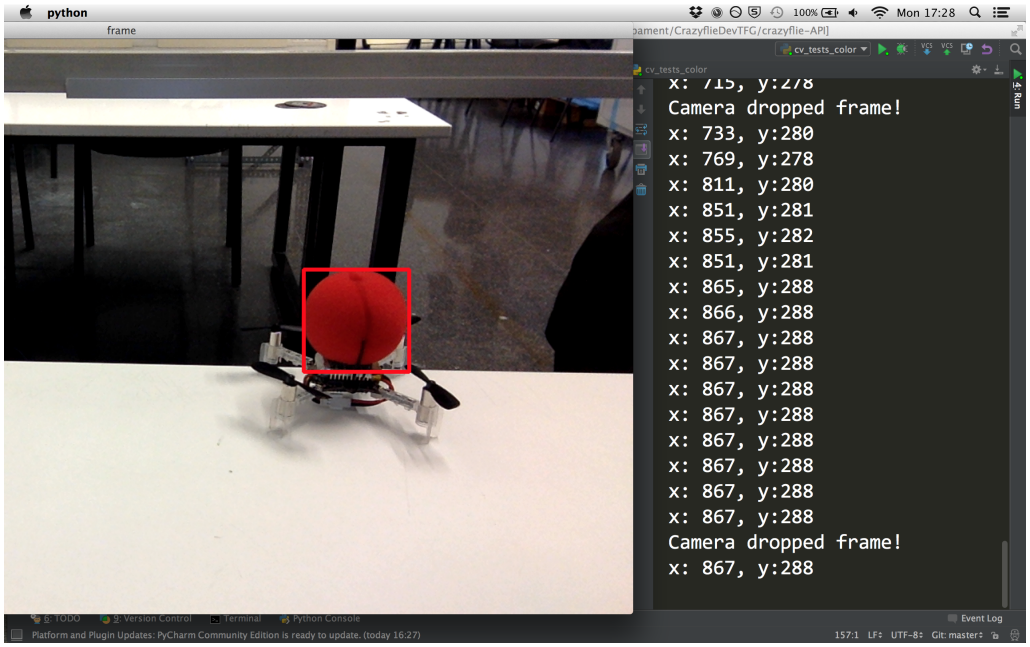


Figura 3.14: Exemple de posicionament mitjançant la detecció del color

Un dels problemes principals que es va observar, a part dels problemes d'estabilitat, és que es perd la dimensió de la profunditat. De la mateixa manera ens passaria si ens tapéssim un ull. Per a poder solucionar aquest problema de profunditat vam trobar varies possibles opcions. Una d'elles és fent ús de marcadors. Coneixent la distància horitzontal podem arribar a estimar la distancia d'un objecte fent una proporcionalitat i entenent els principis bàsics de la perspectiva [24].

D'aquesta manera fent que el *marker* sigui el propi dron i coneixent les seves mesures, podríem arribar a determinar la distància en profunditat a la que està volant l'objecte respecte la càmera.

Una altre opció és fer ús de l'estereografia [25], per exemple usant més d'una càmera. Coneixent la posició exacta de les càmeres (és necessari que estiguin situades horitzontalment i a la mateixa altura) i la relativa entre elles, es podria determinar la distància al punt de referència fent uns càlculs matemàtics posteriors a les fotografies realitzades amb un error màxim de l'ordre de centímetres. És tracta d'una tècnica usada per representar imatges en tres dimensions (figura 3.15).

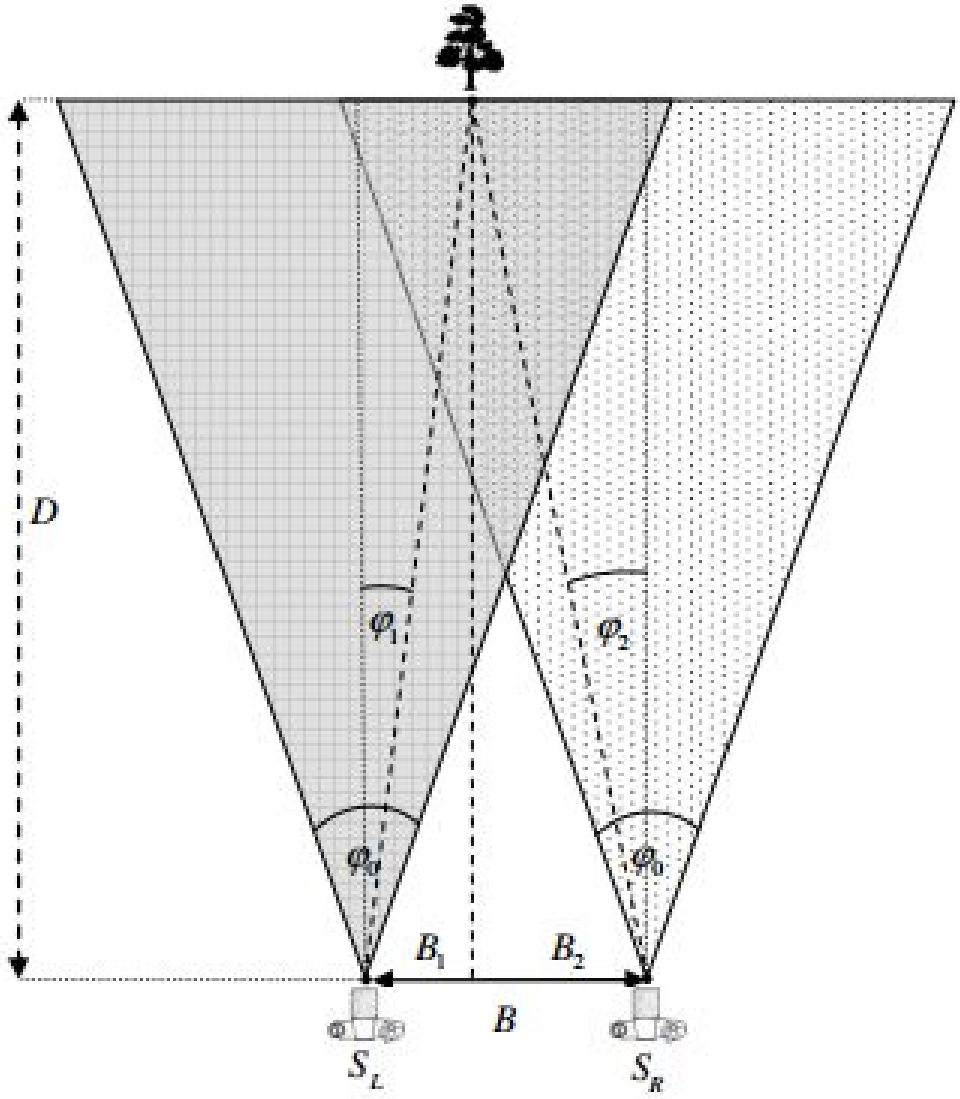


Figura 3.15: Esquemàtic del funcionament de l'estereografia

Un dels principals desavantatges d'aquest mètode és que s'ha de tenir molta precisió a l'hora de col·locar les càmeres així com d'assegurar-se de que aquestes sempre estaran a la mateixa posició i es calibrarà el sistema en funció d'aquestes dades. A causa que els experiments es van dur a terme al llarg d'un període relativament llarg (quatre mesos), i no sempre en el mateix espai, l'opció de l'estereografia va ser descartada. A més a més, en el nostre cas no disposàvem d'una segona càmera amb la que fer les proves.

Per tant es va optar per la primera opció: un sistema de posicionament senzill que dona informació de la posició x, y, z respecte la posició de la càmera per tenir un control limitat del dron dins la imatge que s'obté. Evidentment, i de forma constatada al llarg dels experiments, la limitació més gran que es va patir és, com es porta comentat durant tot l'estudi, l'estabilitat del dron. En el moment en que no som capaços de mantenir el dron quiet en una determinada posició i desapareix de la imatge, es perd el control mitjançant aquest sistema. Més endavant s'analitzarà els avantatges i desavantatges d'aquest sistema primari.

# CAPÍTOL 4. INTEGRACIÓ DELS SISTEMES

En aquest apartat es veurà finalment la integració de tots els sistemes o mòduls anteriorment esmentats i com interactuen entre ells. La integració bàsica serà de dos components: per una banda el sistema de control del Crazyflie i per l'altre el sistema de posicionament.

L'esquema d'interacció dels components es pot descriure amb la següent imatge (figura 4.1):

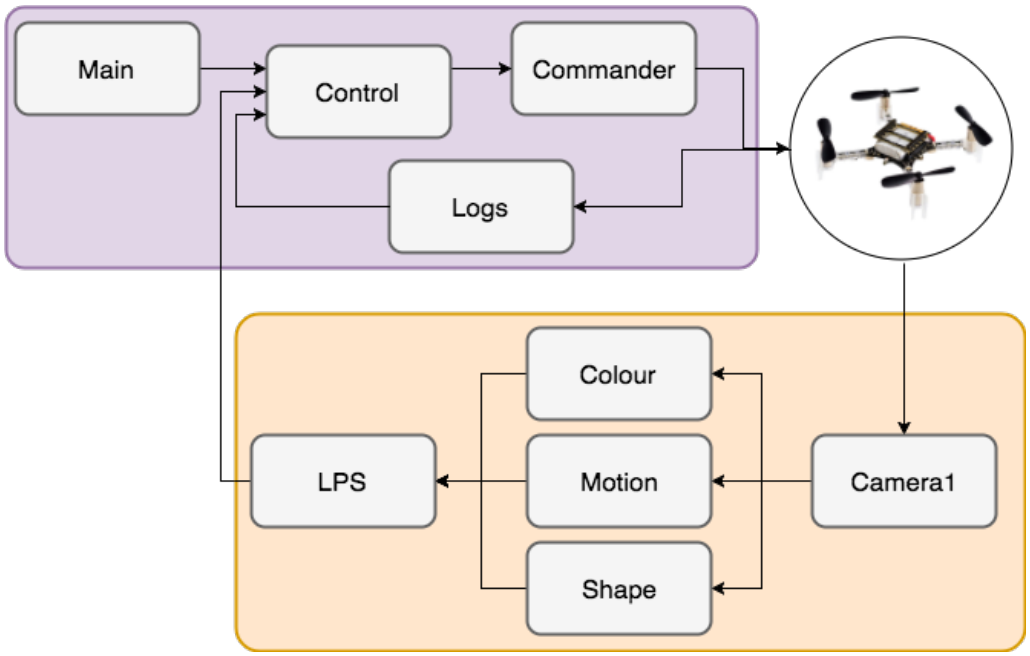


Figura 4.1: Esquemàtic de control i posicionament

D'una banda tenim el sistema de control, on no hi ha canvis significatius respecte l'explicat anteriorment en cap dels sub-mòduls, excepte en el de control, que ara haurà de tenir en compte no només l'*input* de l'usuari i els *logs* de la estabilització sinó també la informació que li arribarà en forma d'*input* que rebrà del sistema de posicionament.

D'altra banda tenim el sistema de posicionament on hi ha la càmera, els sistemes de detecció de color, forma i/o moviment i el nucli principal del sistema on s'obté la posició exacta del Crazyflie que s'envia al control principal del Crazyflie.

D'aquesta manera i segons la descripció anterior, s'ha assumit que la integració dels sistemes amb els que havíem de treballar i experimentar que són els de control i posicionament, s'ha pogut realitzar amb èxit. Per tant només queda realitzar els experiments pertinents per tal de valorar les dades extretes segons la caracterització feta dels mòduls.

## 4.1. Descripció del sistema

En ordre seqüencial, aquest sistema funciona de la següent manera :

1. S'inicien els *drivers* pertinents vist en la secció 2.2.4.

2. Es selecciona l'adreça radio depenent de quin dron es farà servir.
3. Es busquen els canals disponibles i es selecciona un d'aquests.
4. S'activa la classe creada *ControllerCrazyflie* que conté tot el sistema de control.
   (a) Aquesta inicia la connexió amb el Crazyflie
   (b) Configura i comença els *logs* d'acceleració
   (c) Configura i comença els *logs* d'estabilització
   (d) Comença el `commander_thread`
   (e) Comença el `control_thread`, que té inclòs en el sistema l'entrada dels *logs* i també la posició actual rebuda per la càmera.
   (f) S'inicia l'enlairament
5. S'activa la classe anomenada `ImageCapturer` que conté el sistema de detecció del Crazyflie.
   (a) Inicia la càmera
   (b) Defineix l'àrea d'interès on voldrem que el dron es situï
   (c) Es defineix quin tipus de detecció s'usarà, color, formes o moviment
6. Finalment s'entra en un *loop* en què:
   (a) Es captura el *frame* de la càmera i es fa el processament de la imatge
   (b) S'actualitza la informació de la posició actual del Crazyflie
   (c) S'actualitza la gràfica de l'acceleració vertical

A continuació podem observar el codi que executa aquestes comandes del programa principal que integra els dos sistemes, tant el de control com el de posicionament.

```
import cflib.crtp
from controller_cf import ControllerCrazyflie
import lib_capturer as lc
from lib_plots import *

# Init the drivers
cflib.crtp.init_drivers(enable_debug_driver=False)
# Select the radio
address_radio = 0xE7E7E7E7E5
# Scan for radio interfaer
available = cflib.crtp.scan_interfaces(address=address_radio)

if len(available) > 0:
    link_radio = available[0][0]
else: # in case no channels were found
    link_radio = ""
    quit()

# cc meaning crazyflie controller
cc = ControllerCrazyflie(link_radio)
# ic meaning image caputurer
ic = lc.ImageCapturer()
```

```
23
24 while cc.is_connected:
25     # capture actual image
26     ic.start_capturing()
27     # update actual cc position
28     cc.actual_x, cc.actual_y, cc.width = ic.return_x_y()
29     # plot acc_z
30     plot_data(cc.acc_z)
31
32 # if the cc is not connected anymore stop the camera
33 ic.stop_capturing()
```

## 4.2. Desenvolupament i experiments de la integració

Seguint el punt anterior, el que cal i s'explicarà en aquest punt, seran els experiments i proves pertinents realitzades amb el Crazyflie i el sistema de posicionament. Els programes que es van realitzar s'observen a la taula 4.1

Taula 4.1: Programes realitzats amb la llibreria de Python

| Nom del programa | Proves | Descripció |
|---|---|---|
| `Basic_integration` | 25 | arrencar el dron, intentar rebre la posicio actual |
| `Point_target_color` | 30 | posicionar el dron en un punt |
| `Area_target_color` | 10 | posicionar el dron en un àrea |
| `Area_target_motion` | 20 | mitjançant detecció moviment |

Primerament varem intentar fer funcionar tots els *threads* mitjançant el `Basic_integration`. Bàsicament el que es pretenia era arrencar el dron i anar actualitzant la posició actual en funció del posicionament rebut per la càmera. Gràcies a aquestes proves vàrem observar l'ordre de creació dels *threads* i com comunicar el sistema de posicionament amb el sistema de control.

Després es va programar el `Point_target_color` que consistia en realitzar un *take off* i que el dron conservés una posició x, y, z concreta i immòbil en el temps (figura 4.2). Per fer-ho simplement es varen aplicar unes condicions al control que corregien la posició en funció de la posició actual i la posició desitjada, mitjançant la comparació entre elles. El que es pretenia al principi era situar el dron en el píxel del mig de la imatge rebuda, la meitat de la distància vertical i horitzontal de la imatge. El que es va fer va ser dividir la imatge rebuda per la càmera i indicar a l'objecte que es mantingués estàtic en aquest punt. Els resultats d'aquestes proves ens varen servir per calibrar el sistema de posicionament i enviar de forma correcta les posicions al sistema de control. La resposta que vàrem obtenir no va ser del tot la esperada. Ens adonàrem que els canvis de comandes eren massa ràpids i el fet de pretendre que el dron es mantingués en un punt en concret, degut als seus problemes d'estabilitat, era impossible. Tot just després d'enlairar el dron s'allunyava més enllà dels límits de la càmera i perdíem el control instantàniament. A més a més, depenent de quin sistema de detecció s'escollís, els temps de resposta eren molt més lents (un o dos segons) i per tant no es corregia la posició en el mateix moment sinó que s'observava un cert retard.

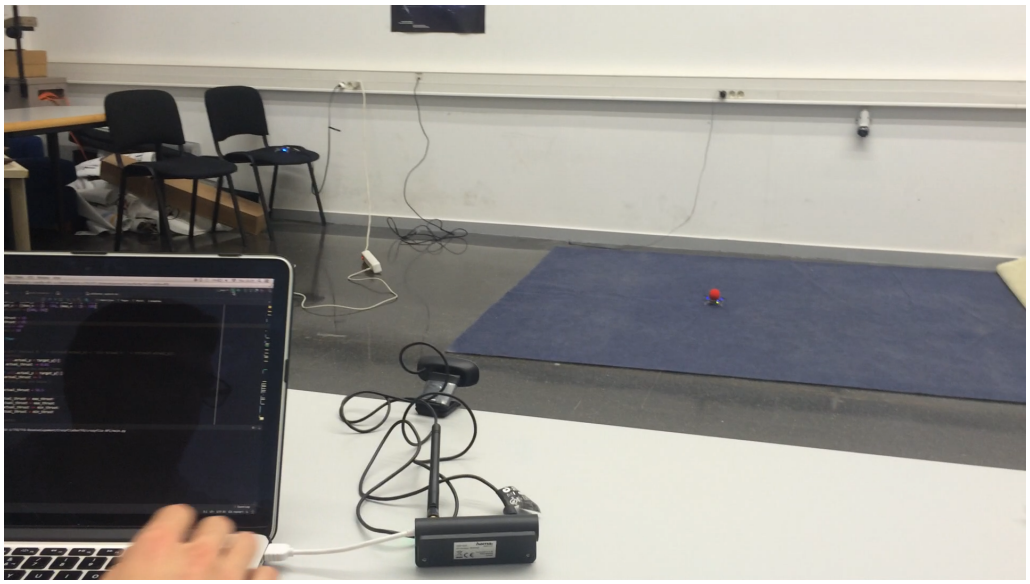

Figura 4.2: Integració dels sistemes

A partir d'aquests resultats, es va optar per definir una àrea rectangular en comptes d'un sol punt amb el programa `Area_target_color` i `Area_target_motion`. Aquesta àrea rectangular va ser primerament dissenyada per situar-se amb el punt mig al centre de la imatge, però finalment es va optar per desplaçar el rectangle lleugerament més a munt del centre per temes relacionats amb l'enfocament de la càmera i de *thrusts* mínims inicials. La idea era bàsicament que, si el dron estava situat dins aquesta àrea, no hi haguessin canvis de comandes, mentre que quan estigués fora d'aquesta àrea definida l'objectiu seria tornar-hi a entrar dins. Usant aquest nou mètode es va poder apreciar notables canvis respecte la idea inicial. D'aquesta forma ja que podíem arribar a posicionar el dron dins l'àrea d'interès (el rectangle) en els moments inicials. Continuava essent difícil ja que tot i implementar les correccions que havia de realitzar el dron una vegada es sortia del rectangle, en el moment en que realitzava aquestes correccions, els moviments es tornaven molt bruscos, es perdia el control del Crazyflie en qüestió de milisegons i es sortia de la imatge. El principal problema que es va patir i que es porta comentant durant tot el document, és que mantenir una certa altitud i estabilitat horitzontal fixa amb el Crazyflie és força complicat. Es va observar com depenent del pes que tingués el Crazyflie, el *thrust* necessari per mantenir-se a una certa alçada canviava. També depenia el *thrust* del nivell de bateria que tingués. Constantment s'havia de calibrar els valors màxims i mínims d'empenta així com variar els valors de control segons el Crazyflie usat (contàvem amb dos exemplars per tal de poder agilitzar la feina) i la situació en que ens trobéssim. Per últim es va intentar implementar una funció feta per desenvolupadors [26][27][28] que es trobava disponible gratuïtament en el Github de Bitcraze que suposadament manté l'altitud constant però no va tenir èxit, no es va poder implementar de forma correcta.

En les següents imatges (figures 4.3 i 4.4) es pot apreciar el Crazyflie dins i fora de l'àrea d'interès així com els diversos mètodes de detecció usats en diferents situacions.

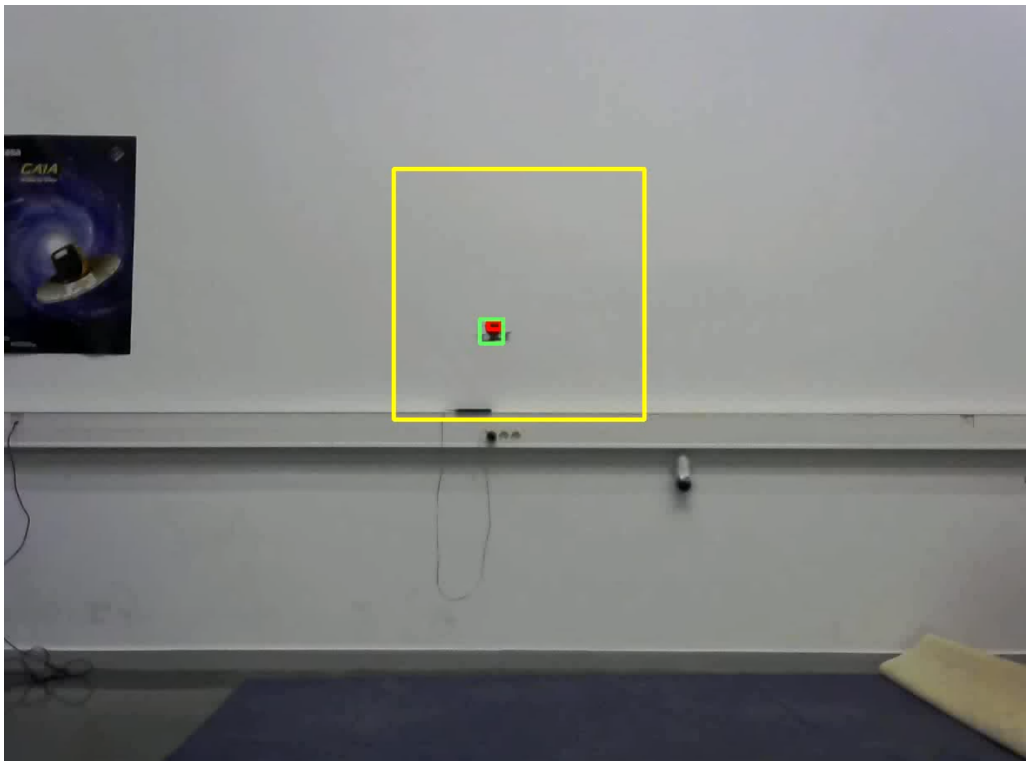

Figura 4.3: Crazyflie dins l'àrea d'interès

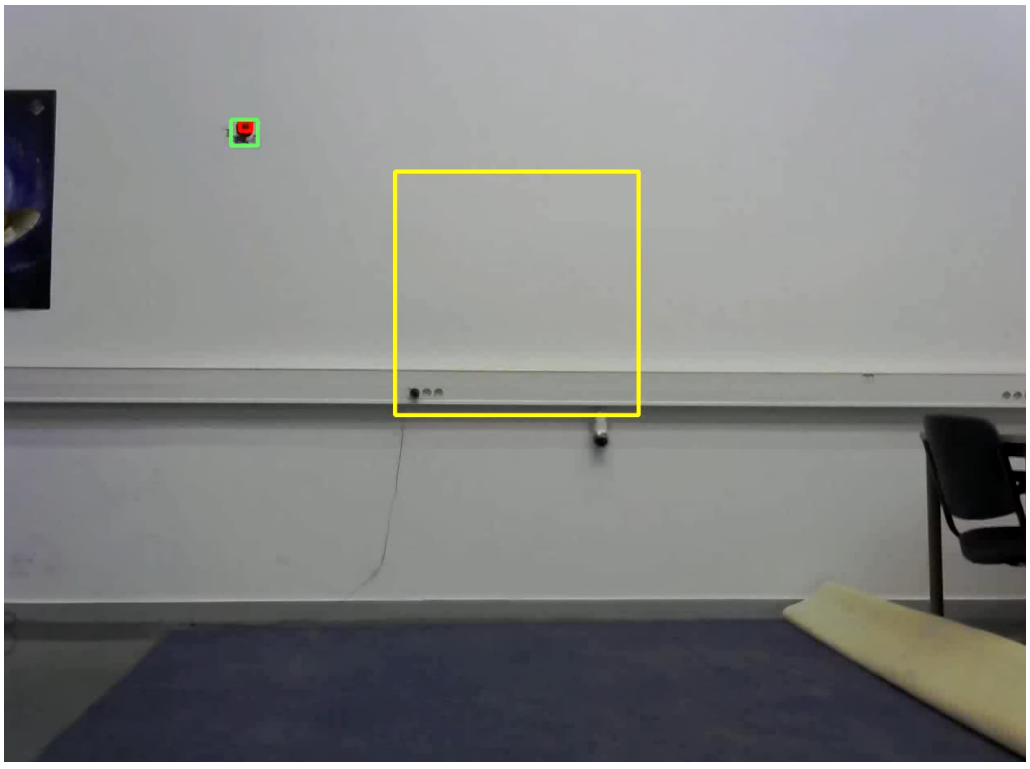

Figura 4.4: Crazyflie fora l'àrea d'interès

# CONCLUSIONS

Amb les tecnologies dron avançant de forma exponencial en els últims anys, s'han presentat noves aplicacions i nous reptes en l'àmbit civil. En gairebé totes les aplicacions que requereixen de gran precisió, o que es desenvolupen en espais tancats i reduïts, la cobertura GPS ha estat una limitació. Mitjançant aquest estudi s'ha observat la necessitat de dotar els sistemes dron amb un sistema de posicionament local que pugui reduir aquestes mancances.

Existeixen diferents tipus de sistemes de posicionament locals on destaquen les tecnologies de radiofreqüència, magnètiques, inercials, làser i la visió per computador. Totes tenen els seus respectius avantatges i inconvenients segons el cost, la implementació i la precisió. El sistema desenvolupat en aquest treball ha estat la visió per computador per la comoditat d'implementació, el cost reduït, i una precisió acceptable de menys de 10 cm.

Per a poder implementar i provar aquest sistema de posicionament local, s'ha escollit la plataforma Crazyflie 2.0 de Bitcraze com a model d'estudi. El Crazyflie és un mini dron de *software open source* (codi obert). Pel seu reduït pes, aquest no disposa d'antena GPS que pugui georeferenciar la seva posició. Per a poder controlar-lo s'ha creat un projecte nou que es basa en la llibreria oberta de Python amb dos mòduls principals: un de control i un de posicionament. S'ha pogut calibrar el dron per a què pugui volar amb un pilot humà. L'estabilitat del dron sense tenir un sistema d'autocontrol és molt precària, sobretot l'estabilitat horitzontal. El control automàtic sense cap tipus de sistema de posicionament o detecció és inviable.

Per a poder implementar un sistema de posicionament remot s'ha intentat fer servir el Parrot AR.Drone 2.0 sense èxit a causa de la impossibilitat d'obtenir les imatges capturades amb la seva càmera frontal. S'ha decidit continuar amb el control del Crazyflie per la seva dificultat afegida malgrat que el sistema de posicionament es podria adaptar senzillament a la plataforma del Parrot.

Per implementar la visió per computador com a sistema de posicionament, s'han desenvolupat tres tècniques de detecció: per colors, forma i moviment. Tan el mètode de detecció per color com el de forma depenien de l'entorn on es feien els experiments. En canvi, en la detecció de moviment l'única limitació era aconseguir que la càmera estigués estàtica. La combinació dels mètodes no ha estat satisfactòria i ha suposat una latència mínima en el sistema de l'ordre de segons. Mitjançant la detecció de moviment s'ha pogut arribar a determinar la posició del Crazyflie en horitzontal i vertical. També s'ha pogut fer una estimació de la profunditat a la que es troba el dron fent una interpolació entre el tamany del dron i els píxels de la imatge.

La integració del sistema de control amb el sistema de posicionament ha estat un èxit. S'han pogut ajuntar el dos sistemes de forma senzilla. Aquesta proposta de disseny fa que el conjunt del sistema sigui més escalable, és a dir, que per exemple es rebi informació de varies càmeres. A més a més, és un sistema adaptable ja que es podria exportar el sistema de posicionament estudiat a altres plataformes dron.

Si bé la integració ha estat un succés, arribar a controlar el dron mitjançant el sistema no ho ha estat. El problema principal ha estat el temps de resposta. En intentar corregir la posició actual del dron, aquest no responia suficientment ràpid i la correcció arribava quan ja era massa tard o ja s'havia corregit. Les possibles solucions per mitigar aquest

tipus de problemes serien per una banda reduir el temps de processament del sistema de posicionament o bé introduir algoritmes per millorar el càlcul de la posició com per exemple el filtre de Kalman.

No s'ha aconseguit per tant, controlar el Crazyflie en una posició estàtica. El posicionament mitjançant visió per computador sí que ha estat possible però amb certes limitacions, com per exemple el temps de resposta ja esmentat anteriorment. També la necessitat de tenir la càmera estabilitzada i calibrada amb uns certs paràmetres no aporta flexibilitat en la localització dels experiments. El fet d'usar només una càmera també suposa que, si per algun motiu el dron s'escapa de l'àrea de visió d'aquesta i per tant de la imatge obtinguda, aquest deixa de ser controlable de forma automàtica ja que es perd la detecció i per tant el posicionament.

# BIBLIOGRAFIA


[1] Eduardo Martos-Naya, Ernesto Marín-Gostrotiza, José Luis Lázaro-Galilea, David Salido-Monzú, Andreas Wieser “”. *Infrared Local Positioning System using Phase Differences*. (IEEE. Texas. 2014): Nov 20-21. 4

[2] Seyed A. (Reza) Zekavat, R. Michael Buehrer “Chapter 1”. *Handbook of Position Location: Theory, Practice, and Advances*. (6 Sep 2014) 4

[3] https://www.nasa.gov/directorates/heo/scan/communications/policy/policy_pnt.html “”. *Global Positioning System History*. [visitat el 06/10/2016] 3

[4] Greg Sterling Senior Analyst, Internet2Go “The Arriva of Indoor GPS”. *Magnetic Positioning*. (June 2014) Opus Research, Inc. 7

[5] Kyle Wroble “Performance Analysis of Magnetic Indoor Local Positioning System”. *Maaster's theses*. (June 2015) Western Michigan University. 7

[6] Chris Anderson “Radio-based position tracking works indoors and out”. . (November 2012) diydrones.com 6

[7] Philipp Vorst, Jürgen Sommer, Christian Hoene, Patrick Schneider, Christian Weiss, Timo Schairer, Wolfgang Rosenstiel, Andreas Zell, Georg Carle “Indoor Positioning via Three Different RF Technologies”. *Computer Science Department*. (2008) University of Tübingen, Tübingen, Germany 6

[8] Ankit Kalbande “Laser-based, Low Cost and High Accuracy Local Positioning System (LPS)”. *Electronics Engineering Department*. (January 2015) Visvesvaraya National Institute of Technology 8

[9] Rommel E. Mandapat “Types of Positioning Systems”. *Development and Evaluation of Positioning Systems for Autonomous Vehicle Navigation*. (2001) University of Florida. 5

[10] Dominic Jud, Andreas Michel “An overview of motion tracking methods”. *Motion Tracking Systems*. (Spring Term 2011) Swiss Federal Institute of Technology Zurich. 4

[11] https://www.bitcraze.io/crazyflie-2/ [visitat el 22/09/2016] 12

[12] https://github.com/bitcraze [visitat el 25/09/2016] 18

[13] Benoit Landry “Planning and Control for Quadrotor Flight through Cluttered Enviroments”. *Deparment of Electrical Engineering and Computer Science*. (June 2015) Massachusetts Institute of Technology 13

[14] Samer Abdelmoeti “Robust Control of an Unmaned Aerial Vehicle with Collision Avoidance”. *Introduction Project*. (April 2016) University of Twente 13

[15] https://www.python.org/ [visitat el 25/09/2016] 19

[16] https://wiki.bitcraze.io/projects:virtualmachine:index [visitat el 15/09/2016] 22

[17] https://github.com/venthur/python-ardrone [visitat el 20/09/2016] 35

[18] http://opencv.org/ [visitat el 10/11/2016] 37

[19] http://opencv-python-tutroals.readthedocs.io/en/latest/py_tutorials/py_imgproc/py_houghcircles/py_hough
[visitat el 20/11/2016] 42

[20] H. K. Yuen, J. Princen, J. Illingworth and J. Kittler “A comparative studi of hough transform methods for circle finding”. *Department of electronics and Electrical Engineering*. (1989) University of Surrey. 42

[21] http://docs.opencv.org/3.0-beta/doc/py_tutorials/py_video/py_bg_subtraction/py_bg_subtraction.html
[visitat el 12/11/2016] 44

[22] http://scikit-learn.org/stable [visitat el 17/12/2016] 46

[23] http://scikit-image.org/ [visitat el 17/12/2016] 46

[24] http://www.pyimagesearch.com/2015/01/19/find-distance-camera-objectmarker-using-python-opencv [visitat el 01/12/2016] 47

[25] Jernen Mrovlje and Damir Vrancic “Distance measuring based on stereoscopic pictures”. *PhD Workshop on Systems and Control*. (October 2008) Izola, Slovenia. 47

[26] https://wiki.bitcraze.io/projects:crazyflie:firmware:altitude_hold [visitat el 25/11/2016] 52

[27] https://github.com/bitcraze/crazyflie-clients-python/issues/239 [visitat el 19/09/2016] 52

[28] https://www.bitcraze.io/2016/03/improved-altitude-hold/ [visitat el 22/10/2016] 52

[29] Sven Roehr, Peter Gulden, Denys Shmakov, Igor Bilous “Wireless Local Positioning”. *Existing Solutions and Novel System Concepts*. (2015) Germany. 9

[30] Kai He, Yueyue Zhang, Yaping Zhu, Weiwei Xia, Ziyan Jia, Lianfeng Shen “A Hybrid Indoor Positioning System Based on UWB and Inertial Navigation”. *School of Information Engineering*. (2015) Southeast University, Nanjing, China. 9

[31] Youngmin Kim, Jae Hyun Seo, Heung Mook Kim “A Local Positioning System Based On VHF Band”.System Research Department. (2016) Daejeon, Korea. 9

# APÈNDIXS

# APÈNDIX A. CODI USAT EN EL SISTEMA INTEGRAT

## A.1. main.py

```
import cflib.crtp
from controller_cf import ControllerCrazyflie
import lib_capturer as lc
from lib_plots import *

# Init the drivers
cflib.crtp.init_drivers(enable_debug_driver=False)
# Select the radio
address_radio = 0xE7E7E7E7E5
# Scan for radio interfaer
available = cflib.crtp.scan_interfaces(address=address_radio)

if len(available) > 0:
    link_radio = available[0][0]
else: # in case no channels were found
    link_radio = ""
    quit()

# cc meaning crazyflie controller
cc = ControllerCrazyflie(link_radio)
# ic meaning image caputurer
ic = lc.ImageCapturer()

while cc.is_connected:
    # capture actual image
    ic.start_capturing()
    # update actual cc position
    cc.actual_x, cc.actual_y, cc.width = ic.return_x_y()
    # plot acc_z
    plot_data(cc.acc_z)

# if the cc is not connected anymore stop the camera
ic.stop_capturing()
```

## A.2. controller_cf.py

```
from __future__ import division
from cflib.crazyflie import Crazyflie
from cflib.crazyflie.log import LogConfig
import time
from threading import Thread
from threading import Timer
import logging


# logging.basicConfig(level=logging.ERROR)


```

```
13
14 class ControllerCrazyflie:
15     def __init__(self, link_uri):
16
17         self.cf = Crazyflie()
18
19         self.cf.connected.add_callback(self.cf_connected)
20         self.cf.disconnected.add_callback(self.cf_disconnected)
21         self.cf.connection_failed.add_callback(self.cf_connection_failed)
22         self.cf.connection_lost.add_callback(self.cf_connection_lost)
23
24         self.roll = -5
25         self.pitch = 5
26         self.yaw = 0
27         self.thrust = 0
28         self.roll_trim = +5
29
30         self.acc_z = 0
31
32         self.stab_pitch = 0
33         self.stab_roll = 0
34         self.stab_yaw = 0
35
36         self.cf.open_link(link_uri)
37         self.is_connected = True
38         self.log_acc = None
39         self.log_control = None
40         self.start_logs()
41         self.fall_safe_activated = False
42         self.thrust_finished = False
43
44         self.actual_x = 0
45         self.actual_y = 0
46         self.actual_z = 0
47
48     def start_logs(self):
49         # config log acceleration
50         self.log_acc = LogConfig(name='Acceleration_log', period_in_ms=200)
51         self.log_acc.add_variable('acc.z', 'float')
52         self.log_acc.data_received_cb.add_callback(self.log_acc_data_received)
53         self.log_acc.error_cb.add_callback(self.log_acc_error_received)
54         # config log stabilizer
55         self.log_control = LogConfig(name='Control_log', period_in_ms=50)
56         self.log_control.add_variable('stabilizer.roll', 'float')
57         self.log_control.add_variable('stabilizer.pitch', 'float')
58         self.log_control.add_variable('stabilizer.yaw', 'float')
59         self.log_control.data_received_cb.add_callback(self.log_control_data)
60         self.log_control.error_cb.add_callback(self.log_control_error)
61
62     def cf_connected(self, link_uri):
63         # First add the logs to cf comparing the TOC
64         self.cf.log.add_config(self.log_acc)
65         self.log_acc.start()
66         self.cf.log.add_config(self.log_control)
67         self.log_control.start()
68
69         # Start the commander thread
70         self.thread_commander = Thread(target=self.master_commander)
```

```
71          self.thread_commander.start()
72
73          # Start the control thread
74          self.thread_control = Thread(target=self.control_from_opencv)
75          self.thread_control.start()
76
77          # then init take off
78          self.take_off()
79
80      def cf_connection_failed(self, link_uri, msg):
81          self.is_connected = False
82
83      def cf_connection_lost(self, link_uri, msg):
84          self.is_connected = False
85
86      def cf_disconnected(self, link_uri):
87          self.is_connected = False
88
89      def log_control_data(self, timestamp, data, logconf):
90          self.stab_pitch = data['stabilizer.pitch']
91          self.stab_roll = data['stabilizer.roll']
92          self.stab_yaw = data['stabilizer.yaw']
93
94      def log_control_error(self, logconf, msg):
95          print('Error when logging %s: %s' % (logconf.name, msg))
96
97      def log_acc_data_received(self, timestamp, data, logconf):
98          self.acc_z = data['acc.z']
99          if self.acc_z < 1.1 and self.fall_safe_activated is False:
100             self.fall_safe_activated = True
101             self.fall_safe()
102
103     def log_acc_error_received(self, logconf, msg):
104         print('Error when logging %s: %s' % (logconf.name, msg))
105
106     def fall_safe(self):
107         self.land()
108         self.fall_safe_activated = False
109
110     def set_thrust(self, percentage):
111         temp_thrust = int((percentage / 100) * 65535)
112         self.thrust = temp_thrust
113
114     def get_thrust(self):
115         temp_thrust = int((self.thrust * 100) / 65535)
116         return temp_thrust
117
118     def take_off(self):
119         self.set_thrust(60)
120         time.sleep(1)
121         self.set_thrust(57)
122         self.thrust_finished = True
123         time.sleep(1)
124
125     def land(self):
126         ramp_thrust = 58
127         while ramp_thrust >= 30:
128             ramp_thrust -= 5
```

```
129             self.set_thrust(ramp_thrust)
130             time.sleep(0.1)
131         self.set_thrust(0)
132 
133     def control_from_opencv(self):
134 
135         while self.thrust_finished is False:
136             pass
137 
138         actual_thrust = self.get_thrust()
139         actual_roll = self.roll
140         max_y = 720
141         max_x = 1280
142         target_y = [(max_y / 2) - 150, (max_y / 2) + 150]
143         target_x = [(max_x / 2) - 150, (max_x / 2) + 150]
144         target_width = [100, 200]
145 
146         min_thrust = 56
147         max_thrust = 58
148         min_roll = -10
149         max_roll = 10
150 
151         while True:
152             if not self.fall_safe_activated:
153                 # print('\tActual X: ' + str(self.actual_x) + '\t\t Actual Y: '
    + str(self.actual_y))
154                 #  Vertical
155                 if self.actual_y < target_y[0]:
156                     actual_thrust -= 0.01
157                     # print('you are above')rome
158                 elif self.actual_y > target_y[1]:
159                     actual_thrust += 5
160                     # print('you are below')
161                 else:
162                     actual_thrust = 56.5
163 
164                 if actual_thrust > max_thrust:
165                     actual_thrust = max_thrust
166                 elif actual_thrust <= min_thrust:
167                     actual_thrust = min_thrust
168 
169                 # Horizontal
170                 roll = self.stab_roll
171                 if self.actual_x < target_x[0]:
172                     actual_roll = roll - 6 + self.roll_trim
173                     # print('you are right')
174                 elif self.actual_x > target_x[1]:
175                     actual_roll = roll + 4 + self.roll_trim
176                     # print('you are left')
177                 else:
178                     actual_roll = roll + self.roll_trim
179 
180 
181                 value_rate = 2
182                 pitch_trim = -5
183                 roll_trim = 0
184 
185                 if self.stab_pitch > pitch_trim:
```

```
                    self.pitch -= value_rate
                elif self.stab_pitch < pitch_trim:
                    self.pitch += value_rate

                if self.stab_roll > roll_trim:
                    self.roll -= value_rate
                elif self.stab_roll < roll_trim:
                    self.roll += value_rate


                self.set_thrust(actual_thrust)
                self.roll = actual_roll

    def master_commander(self):
        print("oooo In the master_commander thread")
        self.cf.commander.send_setpoint(0, 0, 0, 0)

        while True:
            self.cf.commander.send_setpoint(self.roll, self.stab_pitch + self.pitch, self.yaw, self.thrust)
            time.sleep(0.1)

    def cf_shut_down(self):
        self.cf.close_link()
        self.is_connected = False
        print('Crazyflie shut down')
```

## A.3. lib_capturer.py

```
import cv2
import numpy as np
import datetime


class ImageCapturer:
    def __init__(self):
        self.cap = cv2.VideoCapture(0)
        self.fgbg = cv2.createBackgroundSubtractorMOG2()
        self.frame = None
        self.x = 0
        self.y = 0
        max_y = 720
        max_x = 1280
        buffer = 150
        self.point1 = ((max_x / 2) - buffer, (max_y / 2) - buffer)
        self.point2 = ((max_x / 2) + buffer, (max_y / 2) + buffer)

        # VIDEO RECORD
        # Get the width and height of frame
        width = int(self.cap.get(cv2.CAP_PROP_FRAME_WIDTH) + 0.5)
        height = int(self.cap.get(cv2.CAP_PROP_FRAME_HEIGHT) + 0.5)

        # Define the codec and create VideoWriter object
        fourcc = cv2.VideoWriter_fourcc(*'mp4v')  # Be sure to use the lower case
        now = datetime.datetime.now()
        video_name = "tests/" + now.strftime("%Y-%m-%d %H-%M") + ".mp4"
```

```
28          self.out = cv2.VideoWriter(video_name, fourcc, 5, (width, height))
29
30      def color_filter(self):
31          ## Color filtering
32          hsv = cv2.cvtColor(self.frame, cv2.COLOR_BGR2HSV)
33
34          # Lab Calibration
35          lower_red = np.array([0, 100, 75])
36          upper_red = np.array([15, 255, 255])
37
38          # # Home Calibration
39          # lower_red = np.array([0, 150, 100])
40          # upper_red = np.array([5, 255, 255])
41
42          mask2 = cv2.inRange(hsv, lower_red, upper_red)
43          cv2.imshow('mask2', mask2)
44
45          # Get rid of background noise using erosion and fill in the holes using
     dilation and
46          # erode the final image on last time
47          element = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
48          mask = cv2.erode(mask2, element, iterations=2)
49          mask = cv2.dilate(mask, element, iterations=2)
50          mask = cv2.erode(mask, element)
51
52          cv2.imshow('mask', mask)
53
54          # Create Contours for all blue objects
55          _, contours, hierarchy = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.
    CHAIN_APPROX_SIMPLE)
56          maximumArea = 0
57          bestContour = None
58          for contour in contours:
59              currentArea = cv2.contourArea(contour)
60              if currentArea > maximumArea:
61                  bestContour = contour
62                  maximumArea = currentArea
63                  # Create a bounding box around the biggest blue object
64          if bestContour is not None:
65              x, y, w, h = cv2.boundingRect(bestContour)
66              # cv2.rectangle(frame, (x, y), (x + w, y + h), (0, 0, 255), 3)
67          else:
68              x = 0
69              y = 0
70              w = 0
71              h = 0
72
73          return x, y, w, h
74
75      def circle_filter(self):
76          img = cv2.medianBlur(self.frame, 5)
77          gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
78
79          # detect circles in the image
80          circles = cv2.HoughCircles(gray, cv2.HOUGH_GRADIENT, 1.1, 200,
81                                     param1=22, param2=50, minRadius=50,
    maxRadius=100)
82
```

```
83          if circles is not None:
84              # convert the (x, y) coordinates and radius of the circles to
    integers
85              circles = np.round(circles[0, :]).astype("int")
86              return circles
87
88      def motion_filter(self):
89          fgmask = self.fgbg.apply(self.frame)
90
91          # Get rid of background noise using erosion and fill in the holes using
     dilation and
92          # erode the final image on last time
93          element = cv2.getStructuringElement(cv2.MORPH_RECT, (3, 3))
94          fgmask = cv2.erode(fgmask, element, iterations=2)
95          fgmask = cv2.dilate(fgmask, element, iterations=2)
96          fgmask = cv2.erode(fgmask, element)
97
98          # cv2.imshow('mask', fgmask)
99
100         # Create Contours for all blue objects
101         _, contours, hierarchy = cv2.findContours(fgmask, cv2.RETR_EXTERNAL,
    cv2.CHAIN_APPROX_SIMPLE)
102         maximumArea = 0
103         bestContour = None
104         for contour in contours:
105             currentArea = cv2.contourArea(contour)
106             if currentArea > maximumArea:
107                 bestContour = contour
108                 maximumArea = currentArea
109                 # Create a bounding box around the biggest blue object
110         if bestContour is not None:
111             x, y, w, h = cv2.boundingRect(bestContour)
112             cv2.rectangle(self.frame, (x, y), (x + w, y + h), (0, 100, 255), 3)
113         else:
114             x = 0
115             y = 0
116             w = 0
117             h = 0
118
119         return x, y, w, h
120
121     def start_capturing(self):
122         _, self.frame = self.cap.read()
123
124         x, y, w, h = self.color_filter()
125         #xm, ym, wm, hm = self.motion_filter()
126
127         self.x = x
128         self.y = y
129         #self.width = wm
130
131         cv2.rectangle(self.frame, (x, y), (x + w, y + h), (0, 0, 255), 3)
132         #cv2.rectangle(self.frame, (xm, ym), (xm + wm, ym + hm), (100, 255,
    100), 3)
133         #cv2.rectangle(self.frame, self.point1, self.point2, (0, 255, 255), 3)
134
135         cv2.imshow('frame', self.frame)
136
```

```
137         # write the flipped frame
138         self.out.write(self.frame)
139 
140     def return_x_y(self):
141         return self.x, self.y, self.width
142 
143     def stop_capturing(self):
144         cv2.destroyAllWindows()
145         self.cap.release()
146         self.out.release()
```